PENERAPAN
STEGANOGRAFI LSB (*LEAST SIGNIFICANT BIT)*
MENGGUNAKAN KOMBINASI
DES (*DATA ENCRYPTION STANDARD*)
DAN
KODE HUFFMAN

SKRIPSI

Oleh:

SAIFUL NUR BUDIMAN

0710960017-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2012

UNIVERSITAS BRAWIJAYA

PENERAPAN STEGANOGRAFI LSB (*LEAST SIGNIFICANT BIT)* MENGGUNAKAN KOMBINASI *DES* (*DATA ENCRYPTION STANDARD*) DAN KODE HUFFMAN

SKRIPSI

Oleh:

SAIFUL NUR BUDIMAN

0710960017-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2012

LEMBAR PENGESAHAN SKRIPSI

PENERAPAN
STEGANOGRAFI LSB (*LEAST SIGNIFICANT BIT)*
MENGGUNAKAN KOMBINASI
***DES* (*DATA ENCRYPTION STANDARD*)**
DAN
KODE HUFFMAN

Oleh :
SAIFUL NUR BUDIMAN
0710960017-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 04 Juli 2012
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Edy Santoso, S.Si., M.Kom	**Candra Dewi, S.Kom., M.Sc**
NIP. 197404142003121004	**NIP. 197711142003122001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 196709071992031001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Saiful Nur Budiman**
NIM	**: 0710960017-96**
Jurusan	**: Matematika**
Program studi	**: Ilmu Komputer**
Penulisan skripsi berjudul	**: Penerapan Steganografi LSB (*Least Significant Bit*) Menggunakan Kombinasi DES (*Data Encryption Standard*) Dan Kode Huffman**

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 04 Juli 2012
Yang menyatakan,

Saiful Nur Budiman
NIM. 0710960017

PENERAPAN
STEGANOGRAFI LSB (*LEAST SIGNIFICANT BIT)*
MENGGUNAKAN KOMBINASI
***DES* (*DATA ENCRYPTION STANDARD*)**
DAN
KODE HUFFMAN

ABSTRAK

Kriptografi sering digunakan pada proses pengamanan data digital. Pada beberapa hal, teknik ini belum dapat dikatakan aman karena data yang disandikan dapat diketahui keberadaannya. Sebagai alternatifnya perlu dilakukan proses steganografi untuk menyamarkannya. Data yang disisipkan dengan steganografi terbatas oleh luas dari citra penampungnya. Supaya pesan yang disisipkan lebih banyak maka perlu dikompresi. Dengan mengkombinasikan kompresi, kriptografi dan steganografi akan dihasilkan citra stego yang maksimal dan aman.

Pada penelitian ini, pesan yang disisipkan terdiri dari tiga buah berkas *.txt dan tiga buah citra penampung *.bmp. Masing-masing pesan akan dikompres dengan kode Huffman. Pesan yang telah dienkoding dengan kode Huffman tersebut selanjutnya akan di enkripsi dengan DES untuk diamankan. Cipherteks yang dihasilkannya disisipkan pada masing-masing citra penampung dengan steganografi LSB.

Citra stego yang dihasilkan memiliki rata-rata PSNR 61,87db dan MSE 0,108. Kualitas dari PSNR dan MSE ditentukan oleh luas dari citra penampungnya serta jumlah pesan yang akan disisipkan. Citra stego yang dihasilkan aman pada serangan *cropping* atau penambahan *noise*, tetapi tidak pada *rotation* dan *resize*.

THE IMPLEMENTATION OF STEGANOGRAPHY LSB (LEAST SIGNIFICANT BIT) BY COMBINATION OF DES (DATA ENCRYPTION STANDARD) AND HUFFMAN CODE

ABSTRACT

Cryptography has been used for data secure in same cases. This technique is not secure enough because the presence of encoded data may be known. Alternatively, the steganography is needed for fading the data.The inserted data by steganography is limited by the area of the cover image. Compression, therefore is needed to enlarge the number of inserted data. By combining compression, cryptography, and steganography the stego image will be resulted in maximum number and securely.

In this research, the inserted message has consisted for three files of *.txt and three cover image of *.bmp. Message was compressed by Huffman code. The encoded message then was encrypted by DES for securing the message. The resulted ciphertext was inserted in each cover image by steganography LSB.

The result shows that stego image had PSNR 61,87db and MSE 0,108 in average. The quality of PSNR and MSE was determined by the area of cover image and the number of inserted message. The resulted stego image was secured from cropping and noise addition but it was not secured from rotation and resize.

KATA PENGANTAR

Alhamdulillah, dengan mengucapkan puji syukur kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan skripsi ini. Skripsi yang berjudul "**Penerapan Steganografi LSB (*Least Significant Bit*) Menggunakan Kombinasi DES (*Data Encryption Standard*) Dan Kode Huffman**" merupakan salah satu syarat memperoleh gelar Sarjana Komputer pada Program Studi Ilmu Komputer Jurusan Matematika Fakultas MIPA Universitas Brawijaya.

Tidak dapat dipungkiri bahwa tidak mungkin penulis dapat menyelesaikan skripsi ini tanpa bantuan serta dukungan dari banyak pihak. Untuk itu, dengan ketulusan dan kerendahan hati, penulis menyampaikan ucapan terima kasih sebesar-besarnya kepada :

1. Edy Santoso, S.Si., M.Kom., selaku dosen pembimbing utama yang telah meluangkan waktu untuk memberikan pengarahan dan masukan bagi penulis.
2. Candra Dewi, S.Kom., M.Sc., selaku pembimbing yang telah memberikan bimbingan serta bantuannya.
3. Dr. Abdul Rouf Alghofari, M.Sc., selaku ketua Jurusan Matematika.
4. Segenap bapak dan ibu dosen yang telah mendidik dan mengajar ilmunya kepada penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika Fakultas MIPA Universitas Brawijaya.
5. Segenap staf dan karyawan di Jurusan Matematika Fakultas MIPA Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan proposal skripsi ini.
6. Kedua orang tua, bapak dan ibu serta kakak terima kasih atas semua doa, kasih sayang dan perhatian yang tulus serta dukungan yang telah diberikan.
7. M. Azwar Anas, Ivan Yulfrian, Madya Praditya, Ibnur Rusi, Ade Asti, Dinar Rani, Aprilia Zuliharti, Aprilia Dharma, Amugya Indra, Seviana Dwi H., Irine Alful, Tri Prayudianto, Sandro Diaz V, Agung Herdiyanto serta sahabat-sahabat di Program Studi Ilmu Komputer Fakultas MIPA Universitas Brawijaya yang telah banyak memberikan dukungan demi kelancaran pelaksanaan penyusunan skripsi ini.

8. Dan semua pihak yang telah terlibat baik secara langsung maupun tidak langsung yang tidak dapat penulis sebutkan satu per satu terima kasih atas semua bantuan yang telah diberikan.

Semoga skripsi ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan, oleh karena itu dengan segala kerendahan hati, penulis mengharapkan kritik dan saran yang membangun dari pembaca. Kritik dan saran dapat dikirimkan ke emali di sync458r@gmail.com.

Malang, Juli 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR *SOURCE CODE*

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pertukaran informasi dan komunikasi secara mudah dapat dilakukan seiring berkembangnya teknologi digital. Tuntutan akan keamanan sangat diperlukan jika informasi tersebut bersifat privasi dan rahasia. Departemen pertahanan, perusahaan, atau individu tertentu tidak menginginkan informasi rahasia yang disampaikannya diketahui oleh pihak-pihak yang tidak bertanggung jawab. Maka dari itu dikembangkanlah ilmu yang mempelajari tentang keamanan informasi.

Kriptografi merupakan cabang dari matematika yang fokus pada keamanan informasi (Menezes, dkk., 1996). Kriptografi sudah lama digunakan oleh tentara Sparta di Yunani pada permulaan tahun 400 SM. Scytale merupakan alat kriptografi yang digunakan untuk mengirimkan pesan rahasia pada masa itu (Munir, 2004b). Sejalan dengan perkembangannya, kriptografi mulai banyak digunakan untuk mengamankan informasi digital. DES (*Data Encryption Standard)* merupakan salah satu algoritma kriptografi simetri yang tergolong jenis *cipher* blok. DES dikembangkan oleh IBM dibawah kepemimpinan W.L. Tuchman pada tahun 1972. DES beroperasi pada ukuran blok 64-bit. Mengenkripsi setiap 64-bit plainteks (*plaintext*) menjadi 64-bit cipherteks (*chipertext*) dengan menggunakan 56-bit kunci internal (*internal key*). Kunci internal dibangkitkan dari kunci eksternal (*external key*) yang panjangnya 64-bit (Tilborg, 2005). DES memiliki tingkat keamanan yang tinggi dikarenakan untuk membongkar kuncinya saja memerlukan 2^{56} kemungkinan kunci.

Walaupun tingkat keamanan DES yang terbilang cukup tinggi, hasil cipherteksnya masih bisa diketahui letak keberadaannya dengan mata manusia. Beda halnya dengan steganografi, hasil keluarannya secara kasat mata mirip sepeti aslinya, namun bila diolah dengan komputer akan jelas terlihat bedanya. Steganografi merupakan ilmu dan seni menyembunyikan pesan rahasia (*hiding*

message) sedemikian hingga keberadaan pesan tidak terdeteksi oleh mata manusia. Algoritma yang sering digunakan pada steganografi adalah LSB (*Least Significant Bit)* karena mudah dalam pengimplementasiannya. Imlementasi LSB pada citra digital membutuhkan dua properti yakni citra penampung (*cover*) dan pesan rahasia (*message*) yang akan disembunyikan (Munir, 2004c). LSB mengganti bit-bit terakhir di dalam segmen citra penampung dengan bit-bit pesan rahasia sehingga menghasilkan citra stego (*stego image*).

Terkadang pesan yang disisipkan adalah pesan rahasia yang cukup panjang. Hal ini akan menjadi masalah pada proses steganografi LSB apabila kapasitas tampung citra penampung sangat minim. Jalan keluar yang ditawarkan pun adalah mengganti citra penampungnya dengan ukuran yang lebih besar sehingga akan menghasilkan citra stego dengan ukuran yang besar pula. Dengan ukuran yang besar tersebut, sangat kurang efektiv apabila citra stego tersebut akan ditransfer, karena pasti membutuhkan waktu yang relatif lama. Untuk mengatasi hal tersebut, salah satu caranya adalah dengan melakukan kompresi pada pesan yang akan disisipkan. Kompresi data akan merubah aliran masukan data (*original raw data*) ke bentuk aliran data yang lain yang memiliki ukuran lebih kecil. Dengan kompresi, ukuran data menjadi lebih kecil namun tidak mengurangi isi dari informasi yang dikandungnya. Kompresi Huffman merupakan kompresi dengan sifat *lossless*, tidak membuang sedikit pun informasi yang dimiliki oleh berkas asal sehingga cocok untuk kompresi teks. Dari kompresi Huffman, akan terbentuk kode Huffman yang merepresentasikan hasil dari proses kompresinya (Salomon dan Motta, 2010). Semakin banyak jumlah karakter yang sering digunakan pada suatu berkas, maka faktor kompresinya akan lebih bagus.

Pada penelitian sebelumnya yang berjudul "Steganografi *Ciphertext* Pada Citra Digital Menggunakan LSB" (Elfirman, 2010) bahwa penyisipan pesan terenkripsi dilanjutkan dengan steganografi merupakan cara baru untuk memenuhi kebutuhan subtansial kerahasiaan informasi. Pada penelitian tersebut sebenarnya dapat divariasikan dengan dengan kompresi untuk memaksimalkan jumlah pesan rahasia. Berdasarkan penelitian yang sebelumnya tersebut, maka dilakukanlah penelitian yang mengkombinasikan kelebihan

dari kriptografi DES, kompresi Huffman, dan steganografi LSB. Metode kriptografi DES digunakan karena mampu memberikan tingkat keamanan yang baik. Penggunaan kode Huffman dikarenakan cocok pada teks, sehingga menghasilkan kompresi yang maksimal. Sedangkan steganografi LSB, digunakan karena mampu menghasilkan mutu citra stego yang tidak jauh berbeda dengan mutu citra penampung. Dengan latar belakang tersebut maka ditentukanlah judul penelitian ini "**Penerapan Steganografi LSB (*Least Significant Bit*) Menggunakan Kombinasi DES (*Data Encryption Standard*) Dan Kode Huffman**".

1.2 Rumusan Masalah

Berdasarkan uraian singkat pada latar belakang, maka dapat dirumuskan permasalahan sebagai berikut :

1. Bagaimana mengimplementasikan kombinasi kriptografi DES, kode Huffman, dan steganografi LSB pada citra bitmap untuk menghasilkan citra stego yang mampu menyimpan pesan secara maksimal dan aman.
2. Bagaimana perbandingan kualitas citra stego dengan citra penampung dilihat dari PSNR dan MSE-nya.
3. Bagaimana ketahanan plainteks pada citra stego jika dilakukan manipulasi citra seperti *rotation*, *cropping*, *noise*, *resize*.

1.3 Batasan Masalah

Untuk mencegah melebarnya masalah yang diteliti ini, diberikan batasan masalah sebagai berikut :

1. Berkas citra penampung yang digunakan adalah citra bitmap 24-bit, serta citra stego yang dihasilkan berupa citra bitmap 24-bit.
2. Ukuran minimun citra penampung adalah 8x3 atau 3x8 piksel.
3. Data yang ingin disisipkan adalah berkas teks (*.txt) atau *string*.

4. Terdapat berkas eksternal yaitu *table.txt* untuk menyimpan tabel data kompresi yang berupa kode ASCII dan kode Huffman.
5. Terdapat berkas *secretkey.ser* untuk menyimpan panjang kompresi dan panjang dari cipherteks.

1.4 Tujuan Penelitian

Tujuan penelitian yang ingin dicapai antara lain adalah :

1. Mengimplementasikan kombinasi kode Huffman, kriptografi DES dan steganografi LSB untuk mengamankan dan memaksimalkan jumlah pesan rahasia.
2. Menghitung dan menganalisa kualitas citra penampung dengan citra stego menggunakan PSNR (*Peak Signal to Noise Ratio*).
3. Menganalisa ketahanan plainteks pada citra stego terhadap manipulasi citra seperti *rotation, cropping, noise, resize*.

1.5 Manfaat Penelitian

Dengan dihasilkannya perangkat lunak yang mengkombinasikan kriptografi DES, kode Huffman, dan steganografi LSB diharapkan mampu menghasilkan citra stego yang dapat menyimpan pesan lebih banyak dan aman. Mutu dari citra stego yang dihasilkannya pun tidak jauh berbeda dengan citra penampungnya, baik untuk ukurannya maupun kualitas citranya.

1.6 Metodologi Penelitian

Skripsi ini disusun berdasarkan metodologi berikut :

1. Studi literatur
   Mempelajari teori enkripsi, kompresi, dan steganografi dari literatur dan artikel dari berbagai sumber.
2. Perancangan dan implementasi perangkat lunak

Merancang sistem aplikasi dan mengimplementasikan hasilnya ke perangkat lunak yang akan dibuat.

3. Uji coba dan analisa hasil
   Hasil keluaran, citra stego dianalisa dan diuji ketahanannya terhadap beberapa manipulasi citra.

1.7 Sistematika Penulisan

Skripsi ini disususn berdasarkan sistematika penulisan sebagai berikut :

BAB I : PENDAHULUAN
Pada bagian bab ini dijelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA
Beberapa dasar teori yang mendukung skripsi dicantumkan dan dibahas pada bab ini.

BAB III : METODOLOGI DAN PERANCANGAN
Membahas penggunaan teori-teori yang dipakai untuk merancang dan implementasikanya ke dalam perangkat lunak yang akan dibuat.

BAB IV : HASIL DAN PEMBAHASAN
Menerangakan proses implementasi dari perangkat lunak yang dihasilakan, mengujinya, dan menganalisa citra stego yang dihasilkan.

BAB V : KESIMPULAN DAN SARAN
Memuat kesimpulan dari pembahasan yang telah dilakukan dan saran yang diharapkan mampu mengembangakan aplikasi perangkat lunak ini agar lebih baik.


UNIVERSITAS BRAWIJAYA

BAB II
TINJAUAN PUSTAKA

2.1 Kriptografi

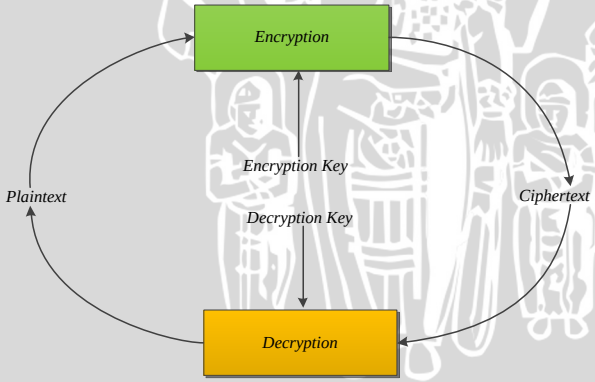
Dalam bahasa Yunani, kriptografi terdiri dari dua macam kata yakni kripto dan graphia. Kripto berarti rahasia, sedangkan graphia berarti tulisan. Jika disatukan makna kriptografi adalah tulisan yang dirahasiakan (Liddell dan Scott, 1996). Secara umum kriptografi merupakan cabang dari ilmu matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, integritas data, keabsahan data, serta otentikasi keaslian data. Selain mengamankan informasi, kriptografi juga merupakan suatu seni tersendiri.

Konsep kriptografi sebenarnya sudah diterapkan manusia sejak 400 SM. Para tentara Sparta dari Yunani menggunakan *Scytale* untuk mengirimkan sebuah pesan rahasia. Bentuk dari *Scytale* berupa pita panjang dari daun papyrus yang dilengkapi dengan silinder. Pesan ditulis secara horizontal baris per baris, apabila pita dilepaskan maka huruf-huruf yang ada didalamnya akan tersusun membentuk pesan rahasia. Untuk membaca pesan rahasia tersebut, penerima harus melilitkan kembali silinder pada diameter sama dengan diameter silinder dari pengirim. Tujuan utama dari pembuatan kriptografi adalah mengamankan pesan yang mencangkup kriteria sebagai berikut (Menezes, dkk., 1996) :

1. Kerahasiaan (*confidentiality*), mengamankan informasi dari semua pihak yang tidak bertanggung jawab. Informasi hanya dapat dibuka oleh beberapa pihak saja yang memiliki kunci rahasia. Kerahasiaan ini erat hubungannya dengan suatu privasi.
2. Integritas data (*data integrity*), menjaga perubahan data yang tidak sah. Untuk menjaga integritas data tersebut, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data yang telah dilakukan oleh pihak yang tidak bertanggung jawab. Manipulasi data meliputi penyisipan, penghapusan, dan pensubsitusian.

3. Otentikasi (*authentication*), berhubungan dengan identifikasi atau pengenalan. Fungsi ini berlaku untuk kedua entitas dan informasi itu sendiri. Dua entitas yang saling berkomunikasi harus mengidentifikasi satu sama lain. Informasi yang disampaikan harus dikonfirmasi sebagai asal, tanggal asal, isi data, dan waktu pengiriman. Untuk alasan inilah aspek kriptografi dibagi menjadi dua kelas utama yaitu otentifikasi entitas dan otentifikasi data asal.
4. Non-repudiasi (*non-repudiation*), digunakan sebagai pencegahan terjadinya penyangkalan terhadap pengiriman.

Enkripsi (*encryption*) disebut juga dengan *encipherment*, merupakan proses mengubah bagian informasi atau plainteks (*plaintext*) ke bentuk sandi rahasia cipherteks (*ciphertext*). Supaya cipherteks dapat dibaca kembali menjadi informasi yang utuh, maka diperlukan proses dekripsi (*decryption*). Dalam proses enkripsi dan dekripsi, dibutuhkan kunci (*key*), dimana kunci tersebut dapat bersifat publik (*public*) atau pribadi (*private*) tergantung algoritma sandi yang digunakan. Pada Gambar 2.1 ditunjukkan illustrasi dari siklus sistem kriptografi.

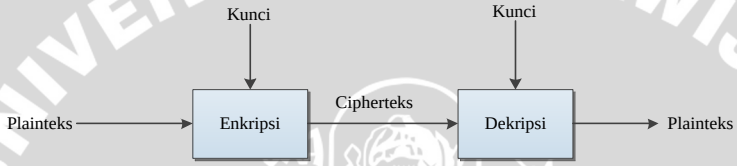


Gambar 2.1 Siklus Sistem Kriptografi (Mao, 2003)

Kekuatan dari algoritma sandi diukur berdasarkan seberapa cepat usaha untuk memecahkan cipherteks menjadi plainteks. Semakin banyak usaha yang diperlukan untuk memecahkannya berarti semakin baik algoritma kriptografi tesebut. Pada sistem

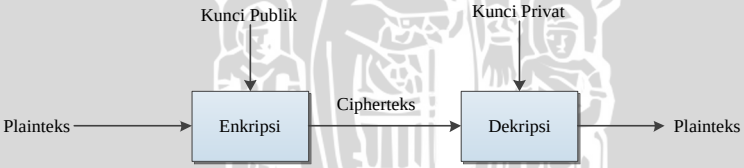
kriptografi modern, kekuatan algoritmanya terletak pada kuncinya (Munir, 2004). Berdasarkan kesamaan kuncinya, algoritma sandi dibagi menjadi dua (Dent dan Mitchell, 2005) :

1. Kunci simetris (*symetric key*), disebut juga algoritma sandi konvensional karena umumnya diterapkan pada algoritma sandi klasik yang menggunakan satu kunci rahasia yang sama untuk melakukan enkripsi maupun dekripsi. Proses enkripsi dan dekripsi kunci simetris ditunjukkan pada gambar 2.2.



Gambar 2.2 Enkripsi dan Dekripsi Kunci Simetris

2. Kunci asimetris (*asimetric key*), menggunakan sepasang kunci yang berbeda dalam enkripsi dan dekripsinya. Kunci tersebut disebut dengan kunci publik (*public key*) dan kunci pribadi (*private key*). Proses enkripsi dan dekripsi kunci asimetris ditunjukkkan pada gambar 2.3.



Gambar 2.3 Enkripsi dan Dekripsi Kunci Asimetris

Pada kunci simetris, berdasarkan jumlah data per proses serta alur pengolahan data didalamnya maka dibedakan menjadi dua kelas yakni :

1. *Cipher* blok (*block cipher*), merupakan skema algoritma sandi yang membagi plainteks yang akan dikirim dengan ukuran tertentu yang disebut blok dengan panjang *t*, dan setiap blok dienkripsi dengan menggunakan kunci yang sama. *Cipher* blok memproses plainteks dengan blok yang

relatif panjang lebih dari 64-bit untuk mempersulit pola serangan dalam membongkar kunci.

2. *Stream cipher*, mengenkripsi pada persatuan data, seperti bit, *byte*, *nible* atau perlima bit. Setiap proses enkripsi menggunakan kunci hasil pembangkitan dari kunci sebelumnya.

2.1.1 DES (*Data Encryption Standard*)

DES, merupakan salah satu anggota algoritma kunci simetris yang dikembangkan oleh IBM dibawah kepemimpinan W.L. Tuchman pada tahun 1972 yang tergolong dalam cipher blok. DES mengenkripsikan 64-bit plainteks menjadi 64-bit cipherteks dengan menggunakan 56-bit kunci internal (*internal key*) atau upa-key (*subkey*). Kunci internal dibangkitkan dari kunci eksternal (*external key*) yang panjangnya 64-bit . Apabila panjang kunci eksternal tidak 64-bit, maka dilakukan penambahan karakter '0'.

2.1.2 Enkripsi

Sebelum melakukan penyandian (*enchipering*), terlebih dahulu dibangkitkan kunci internal berdasarkan kunci eksternalnya. Pada kunci eksternal dilakukan matrik permutasi kompresi *PC-1* yang ditunjukkan pada tabel 2.1.

Table 2.1 *Permutation Choice-1*

PC-1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Dalam permutasi tersebut, setiap bit kedelapan dari tiap *byte* akan diabaikan, sehingga diperoleh matrik kunci internal sepanjang 56-bit. Proses berikutnya membagi kunci internal tersebut menjadi dua bagian yang sama. Setiap bagian masing-masing 28-bit yang disimpan didalam C_0 dan D_0. Kedua bagian baik C_i dan D_i digabung kemudian digeser ke kiri (*left shift*) sepanjang satu atau dua bit bergantung pada tiap putarannya. Jumlah pergeseran pada tiap putaran ditunjukkan pada tabel 2.2.

Table 2.2 *Left Shift*

Putaran ke-*i*	Pergeseran bit
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Pada pergeseran yang ke-16, maka akan diperoleh nilai :

$$\boldsymbol{C_{16} = C_0}$$
$$\boldsymbol{D_{16} = D_0} \qquad (2.1)$$

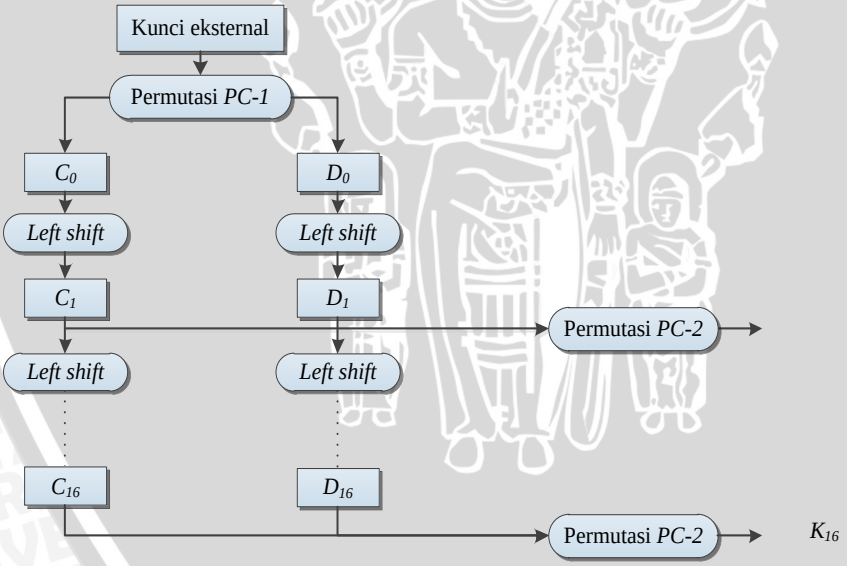
Setelah dilakukan pergeseran ke-*i* (C_i, D_i), selanjutnya dilakukan permutasi kompresi dengan matrik *PC-2* yang ditunjukkan pada tabel 2.3. Dengan pertmutasi ini, maka kunci internal K_i

diturunkan dari (C_i, D_i) sehingga panjang dari setiap kunci K_i akan diperoleh 48-bit.

Table 2.3 *Permutation Choice-2*

PC-2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Secara keseluruhan bagaimana proses dari pembentukan kunci internal DES ditunjukkan pada gambar 2.4.



Gambar 2.4 Pembentukan Kunci Internal

Blok plainteks yang akan diproses harus memenuhi syarat yakni merupakan kelipatan dari 64-bit. Pada perangkat lunak ini, apabila panjang blok plainteks kurang dari syarat tersebut maka diberi imbuhan spasi agar jumlahnya genap menjadi 64-bit. Dilakukan permutasi awal (*IP*) terhadap 64-bit plainteks sesuai tabel 2.4.

Table 2.4 Permutasi Awal

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Dari matrik permutasi awal tersebut dibagi menjadi dua bagian yakni bagian kiri (L_0) dan bagian kanan (R_0). Tiap blok R_0 dan L_0 masing terdiri dari 32-bit yang ditunjukkan pada tabel 2.5 dan 2.6.

Table 2.5 Blok plainteks kiri (L_0)

L_0							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8

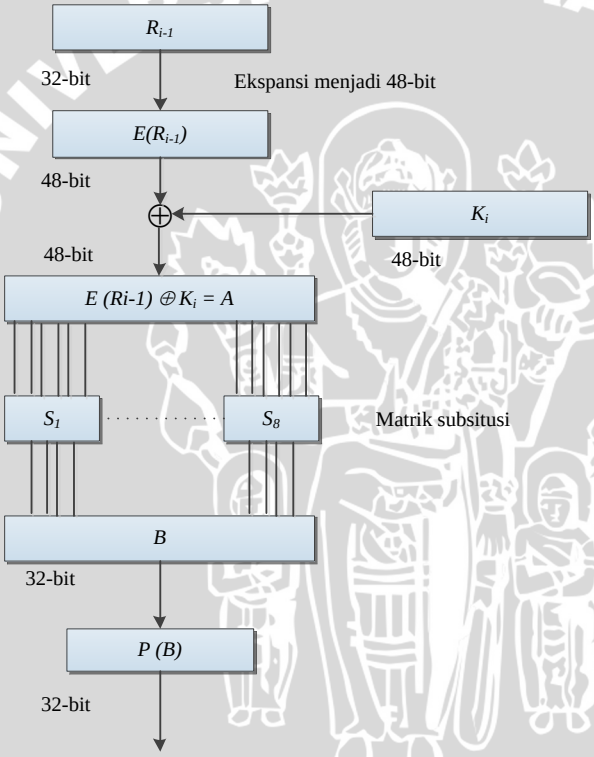
Table 2.6 Blok plainteks kanan (R_0)

R_0							
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Bagian bit-bit plainteks L_0 dan R_0 mengalami perputaran sebanyak 16 kali. Setiap putarannya merupakan jaringan Feistel (*f*) yang ditunjukkan pada persamaan 2.2.

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \quad (2.2)$$

Rincian perhitungan untuk fungsi *f*, ditunjukkan pada gambar 2.5.



Gambar 2.5 Perhitungan Fungsi *f*

Untuk $f(R_{i-1}, K_i)$ yang semula panjangnya adalah 32-bit, diperluas dengan proses ekspansi (*expansion E*) sehingga menjadi 48-bit. Tabel Ekpansi ditunjukan pada tabel 2.7.

Table 2.7 *Expansion*

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Hasil dari $E(R_{i-1})$ yang 48-bit selanjutnya di XOR-kan dengan kunci intenal K_i yang panjangnya 48-bit yang ditunjukkan pada persamaan 2.3. Dari hasil perhitungan tersebut akan diperoleh vektor *A* dengan panjang 48-bit.

$$\boldsymbol{E(R_{i-1}) \oplus K_i = A} \qquad (2.3)$$

Vektor *A* dikelompokkan menjadi 8 kelompok, dimana tiap kelompok memiliki panjang 6-bit. Kelompok 6-bit pertama akan disubsitusikan ke kotak S_1, kelompok 6-bit kedua disubsitusikan ke kotak S_2, begitu seterusnya sampai kelompok 6-bit ke delapan. Pada tiap kotak-*S* terdapat baris dan kolom. Indek baris dibentuk dari bit terdepan dan bit teakhir yang kemudian digabung dan dicari nilai desimalnya. Sisanya yang berupa 4-bit, dicari nilai desimalnya yang akan digunakan sebagai indek kolom. Tabel tiap kotak-*S* dapat ditunjukkan pada tabel 2.8 sampai tabel 2.15.

Table 2.8 Kotak-S_1

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 2.9 Kotak-S_2

S_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Table 2.10 Kotak-S_3

S_3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Table 2.11 Kotak-S_4

S4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Table 2.12 Kotak-S_5

S5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	7	14	2	13	6	15	0	9	10	4	5	3

Table 2.13 Kotak-S_6

S_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Table 2.14 Kotak-S_7

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	3	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	7	5	0	15	14	2	3	12

Table 2.15 Kotak-S_8

S_8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Keluaran dari proses subsitusi pada kotak-*S* menghasilkan vektor *B* dengan panjang 32-bit. Selanjutnya dilakukan permutasi pada vektor *B* dengan matrik permutasi *P* (*P-Box*) yang ditunjukkan pada tabel 2.16. Tujuan dari dilakukan permutasi *P* adalah mengacak hasil proses subsitusi pada kotak-*S*.

Table 2.16 Permutasi *P*

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Dari permutasi *P* terbentuklah bit-bit *P*(*B*) yang merupakan keluaran dari fungsi *f*. Bit-bit *P*(*B*) di **XOR**-kan dengan L_{i-1} untuk mendapatkan R_i yang baru. Jadi dapat disimpulkan keluaran dari putaran ke-*i* berdasarkan fungsi *f* ditunjukkan pada persamaan 2.4.

$$(L_i , R_i) = (R_{i-1}, L_{i-1} \oplus P(B)) \quad (2.4)$$

Setelah diputar sebanyak 16 kali, maka diperoleh nilai L_{16} dan R_{16}. Langkah terakhir untuk mendapatkan cipherteks adalah melakukan invers permutasi (IP^{-1}) sesuai tabel 2.17 terhadap L_{16} dan R_{16}. Secara keseluruhan proses enkripsi DES ditunjukkan pada gambar 2.6

Table 2.17 Permutasi Akhir

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

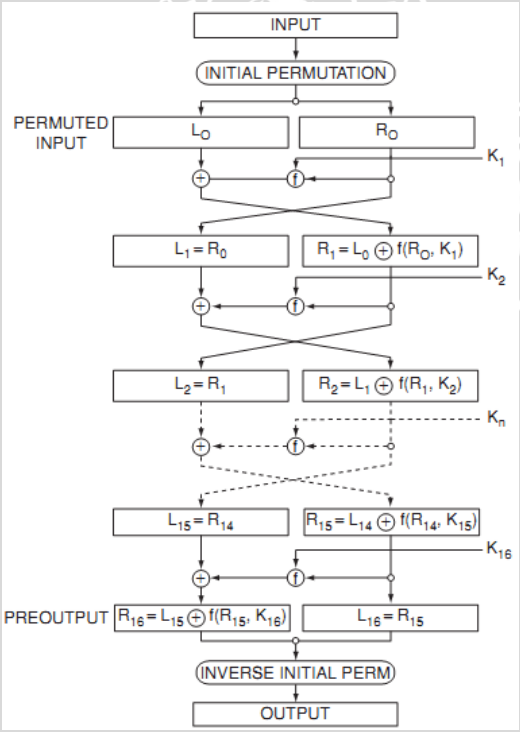
2.1.3 Dekripsi

Untuk memperoleh kembali teks yang sudah menjadi cipherteks tadi, maka diperlukan proses dekripsi. Proses dekripsi terhadap cipherteks merupakan kebalikan dari proses enkripsi. Jika proses enkripsi urutan kunci internal yang digunakan adalah K_1, K_2, K_3 ....K_{16} maka pada proses dekripsinya menggunakan urutan kunci yang berkebalikan yakni mulai dari K_{16}, K_{15}, K_{14}, ... K_1. Langkah-lahngkah dekripsi DES sebagai berikut (Munir, 2004a) :

1) Untuk tiap putaran 16, 15, 14, ... 1, keluaran pada setiap putran *deciphering* menggunakan persamaan 2.2. Dalam hal ini (R_{16}, L_{16}) adalah blok masukan awal untuk deciphering. Blok (R_{16}, L_{16}) diperoleh dengan mempermutasikan cipherteks dengan matrik permutasi IP^{-1}. Pra keluaran dari deciphering adalah (L_0, R_0). Dengan permutasi awal *IP* akan didapatkan kembali blok plainteks yang semula.
2) Meninjau kembali proses pembangkitan kunci internal. Selama *deciphering*, K_{16} dihasilkan dari (C_{16}, D_{16}) dengan

permutasi *PC-2*. Tentu saja (C_{16}, D_{16}) tidak dapat diperoleh langsung pada permulaan *deciphering*. Tetapi karena (C_{16}, D_{16}) = (D_0, C_0), maka K_{16} dapat dihasilkan dari (C_0, D_0) tanpa perlu lagi melakukan pergeseran bit-bit dari kunci eksternal *K* yang diberikan pengguna pada waktu dekripsi.

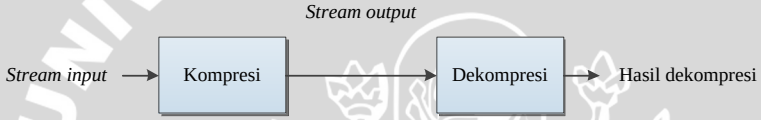
3) Selanjutnya, K_{15} dihasilkan dari (C_{15}, D_{15}) yang mana (C_{15}, D_{15}) diperoleh dengan menggeser C_{16} (yang sama dengan C_0) dan D_{16} (yang sama dengan D_0) satu bit ke kanan. Sisanya, K_{14} sampai K_1 dihasilkan dari (C_{14}, D_{14}) sampai (C_1, D_1). Untuk memperoleh (C_{i-1}, D_{i-1}) diperoleh dengan menggeser C_i dan D_i dengan cara yang sama, tetapi pergeseran kiri (*left shift*) diganti menjadi pergeseran kanan (*right shift*).



Gambar 2.6 Proses Enkripsi DES (Tilborg, 2005)

2.2 Kompresi

Kompresi merupakan bagian dari ilmu komputer yang mempelajari informasi secara terstruktur dengan cara mengubah sekumpulan data menjadi suatu bentuk kode untuk menghemat kebutuhan tempat penyimpanan dan waktu untuk transmisi data. Ini menjadi bagian yang sangat penting dalam perkembangan revolusi *multimedia digital*. Dengan kompresi data dapat dilihat efisiensi dari data seperti teks, gambar, suara, dan video. Tujuan utama dari kompresi data adalah memampatkan dan menyusun kembali data dengan persyaratan rekontruksi yang minimal (Pu, 2006). Ilustrasi dari proses kompresi dan dekompresi ditunjukkan pada gambar 2.7.



Gambar 2.7 Kompresi dan Dekompresi

Sebenarnya konsep utama dalam kompresi terletak pada eliminasi redudansi. Sebagai contoh, dalam kehidupan sehari-hari sering digunakan istilah singkatan dalam penulisan. Penggunakan simbol '&' untuk menggantikan kata 'dan' serta penggunaan kata 'yg' untuk menyingkat kata 'yang'. Dari konsep tersebut, permasalahan pada kompresi data adalah bagaimana menemukan metode yang efisien untuk menghilangkan redudansi dari berbagai tipe data serta metode untuk mengembalikan hasil kompresi ke bentuk semula. Diperlukan suatu algoritma tertentu untuk mengubah data asli menjadi ukuran yang lebih kecil. Secara garis besar terdapat dua macam agloritma kompresi data, yakni (Sayood, 2006) :

1. Kompresi *Lossless*, sifatnya tidak menghilangkan sedikitpun informasi yang dimiliki oleh berkas asal. Algoritma *lossless* ini sering digunakan pada jenis berkas teks, citra medis (*medical image*) dan *binary* (*executable*). Hal ini dikarenakan sedikit perubahan pada berkas yang dikompresi akan memberi pengaruh yang besar pada berkas setelah didekompresi ulang. Sebagai contohnya pada suatu berkas program komputer (*Source code*), sedikit perubahan yang

terjadi akan berakibat pada kesalahan kode program tersebut saat di kompilasi setelah di dekompresi.

2. Kompresi *Lossy*, untuk mendapatkan hasil yang optimum pada berkas yang dimampatkan maka perlu membuang beberapa bagian informasi dari berkas tersebut. Algoritma yang bersifat *lossy* sering digunakan untuk memampatkan berkas citra multimedia, *video*, atau suara. Dengan sistem penglihatan dan pendengaran manusia yang terbatas, beberapa detail informasi dari berkas citra multimedia, *video*, dan suara dapat dihilangkan sehingga hasil kompresi seolah-olah mirip seperti data aslinya.

Berdasarkan tipe peta kode yang digunakan untuk mengubah pesan awal menjadi sekumpulan *codeword*, maka kompresi dibagi menjadi :

1) Metode statik, menggunakan peta kode yang selalu sama. Metode ini membutuhkan dua fase (*two- pass*). Fase pertama untuk menghitung probabilitas kemunculan tiap simbol atau karakter kemudian menentukan peta kodenya. Fase kedua untuk mengubah pesan menjadi kumpulan kode yang akan ditansmisikan.
2) Metode dinamik (*adaptif*), menggunakan peta kode yang dapat berubah dari waktu ke waktu. Metode ini disebut *adaptif* karena peta kode mampu beradaptasi terhadap perubahan karakteristik isi berkas selama proses kompresi berlangsung. Metode ini bersifat satu fase (*one-pass*), karena hanya diperlukan satu kali pembacaan terhadap isi berkas. Contoh dari algoritma ini adalah LZW dan DMC.

Berdasarkan teknik pengkodean atau pengubahan simbol yang digunakan, metode kompresi dibagai menjadi :

1) Metode *symbolwise*, menghitung peluang kemunculan dari tiap simbol dalam berkas masukkan. Kemudian mengkodekan satu simbol dalam satu waktu, dimana simbol yang lebih sering muncul diberi kode lebih pendek dibandingkan simbol yang lebih jarang muncul. Algoritma yang menggunakan *symbolwise* adalah algoritma Huffman.

2) Metode *dictionary*, umumnya membandingkan pola bagian data yang akan diproses dengan bagian data yang sudah diproses sebelumnya. Kemudian menggunakan kode sebagai tanda pengenal yang merujuk pada pola perulangan. Contoh algoritman kompresi yang menggunkan metode ini adalah LZW.
3) Metode *predictive*, menggunakan model *finite-content* atau *finite-state* untuk memprediksi distribusi probabilitas dari simbol-simbol selanjutnya. Contoh algoritmanya adalah DMC.

Faktor kompresi (*compression factor*) merupakan *invers* dari nilai rasio kompresi (*compression ratio*) yang menunjukkan prosentase ukuran berkas hasil pemampatan dibandingkan ukuran berkas sebelum dimampatkan. Pada persamaan 2.5 ditunjukkan proses perhitungan untuk faktor kompresi, sedangkan rasio kompresi ditunjukkan pada persamaan 2.6 (Pu, 2006) .

$$\textbf{Faktor kompresi} = \frac{\textbf{ukuran berkas masukkan}}{\textbf{ukuran berkas keluaran}} \quad (2.5)$$

$$\textbf{Rasio kompresi} = \frac{\textbf{ukuran berkas keluaran}}{\textbf{ukuran berkas masukkan}} \quad (2.6)$$

Terlihat bahwa rasio kompresi akan selalu bernilai kurang dari 1, apabila lebih berarti kompresi terebut jelek (Sayood, 2006). Jika suatu algortima kompresi memiliki nilai rasio kompresi 0,5 maka algoritma tersebut mampu memampatkan berkas hingga setengah ukuran asalnya (50%). Jadi semakin kecil nilai rasio kompresi dari suatu algoritma kompresi maka semakin bagus algoritma tersebut. Nilai faktor kompresi berkebalikan dengan rasio kompresi. Semakin kecil nilai faktor kompresi dari suaru algoritma pemampatan, maka algoritma tersebut semakin buruk. Nilai faktor kompresi dan rasio kompresi lebih sering digunakan sebagi standart tingkat keandalan dari suatu algoritma kompresi.

Masalah yang timbul ketika dilakukan kompresi adalah aspek efisiensi dari algoritma yang digunakan. Secara tidak langsung algoritma dari kompresi bergantung pada data dan struktur internalnya. Ada beberapa faktor yang sering menjadi pertimbangan

dalam memilih suatu algoritma kompresi yang tepat yaitu kecepatan kompresi, sumber daya yang dibutuhkan, ukuran berkas hasil kompresi, besarnya redundansi, serta kompleksitas algoritmanya.

2.2.1 Kompresi Huffman

Algoritma Huffman atau sering disebut kode Huffman adalah salah satu metode kompresi yang paling banyak digunakan untuk memampatkan data (Salomon dan Motta, 2010). Algoritma ini ditemukan oleh David Huffman pada tahun 1952, seorang mahasiswa MIT. Jenis kompresinya bersifat *lossless* sehingga isi dari informasi yang dikompres masih tetap terjaga. Berdasarkan tipe kode yang digunakan pada algoritma Huffman menggunakan metode statik. Sedangkan untuk teknik pengkodeannya menggunakan *symbolwise*.

2.2.2 Kompresi

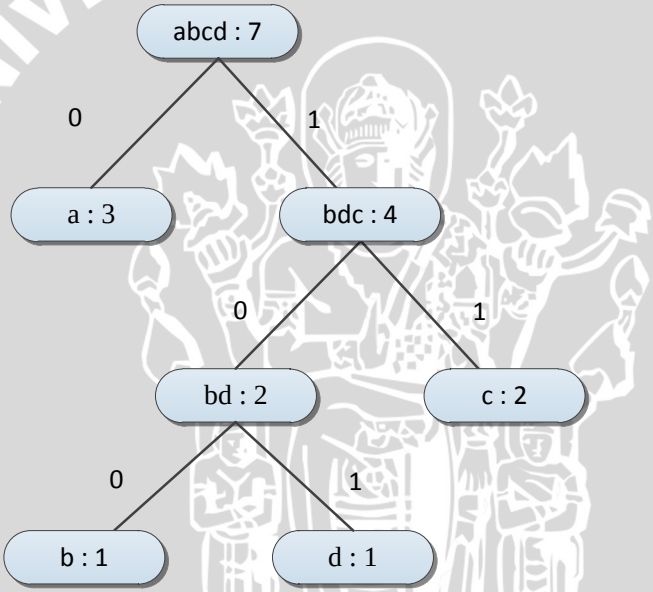
Dalam proses kompresinya, pembentukan kode dengan cara memperhatikan frekuensi kemunculan tiap simbol tertentu pada data yang diolah. Selanjutnya membuat tabel frekuensi dan peluang dari setiap simbol. Apabila tabel sudah terbentuk, barulah membuat pohon Huffman. Dalam pembentukan pohon Huffman, *node*-nya berupa simbol serta frekuensi kemunculannya. Pada *node* sebelah kiri diberi label '0' dan *node* sebelah kanan diberi label '1'. Pada percabangan sebelah kanan dilakukan proses perulangan sampai seluruh simbol habis. Serangkaian proses tersebut akan menghasilkan bit-bit biner yang merupakan kode awalan (*prefix code*). Semakin sering frekuensi suatu simbol muncul, maka memiliki jumlah kode awalan yang sedikit begitu pula sebaliknya.

Sebagai contoh perhitungan dalam kompresi Huffman, dimasukkan teks '**abaccda**' . Tabel 2.18 merupakan tabel frekuensi dan peluang dari tiap simbol berdasarkan masukkan teks tersebut.

Table 2.18 Frekuensi dan Peluang Simbol

Simbol	Frekuensi	Peluang
a	3	3/7
b	1	1/7
c	2	2/7
d	1	1/7

Setelah tabel terbentuk, langkah selanjutnya adalah membuat pohon Huffman yang digambarkan pada gambar 2.8.



Gambar 2.8 Pohon Huffman

Pembentukan pohon Huffman dimulai dari node yang memiliki peluang terkecil terlebih dahulu. Berikutnya jumlah kedua peluang dari node tesebut dijumlah, dan dibandingkan lagi dengan node-node yang lainnya. Proses tersebut diulangi sampai terbentuk satu buah pohon Huffman. Berdasarkan gambar pohon Huffman pada gambar 2.8, maka hasil kode Huffman-nya ditunjukkan pada tabel 2.19.

Table 2.19 Kode Huffman

Simbol	Frekuensi	Peluang	Kode Huffman
a	3	3/7	0
b	1	1/7	100
c	2	2/7	11
d	1	1/7	101

Perbandingan jumlah bit sebelum dan sesudah kompresi ditunjukkan pada tabel 2.20.

Table 2.20 Perbandingan Hasil Kompresi Kode Huffman

Simbol	Desimal	Biner	Kode Huffman
a	97	01100001	0
b	98	01100010	100
a	97	01100001	0
c	99	01100011	11
c	99	01100011	11
d	100	01100100	101
a	97	01100001	0
Total		**56-bit**	**13-bit**

Terlihat bahwa jumlah bit-bit setelah dikompresi dengan kode Huffman adalah 13-bit. Hal ini menunjukkan bahwa ukuran berkas mengalami penurunan secara drastis yakni dari 56-bit menjadi 13-bit.

2.2.3 Dekompresi

Untuk melakukan dekompresi pada kode Huffman, dapat dilakukan dengan dua cara. Cara yang pertama adalah dengan menggunakan tabel kode Huffman dan cara yang kedua adalah menggunakan pohon Huffman. Proses dekompresi berdasarkan pohon Huffman sebagai berikut (Kusdinar, dkk., 2005) :

1. Baca sebuah bit dari *string* biner.
2. Untuk setiap bit pada langkah 1, lakukan pembacaan pada cabang yang bersesuaian.
3. Ulangi langkah 1 dan langkah 2 samapai bertemu dengan node. Kodekan rangkaian yang telah dibaca dengan karakter pada *leaf*.
4. Ulangi dari langkah 1 sampai semua bit di dalam *string* habis.

2.3 Citra Digital

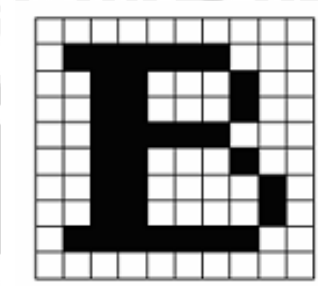
Citra merupakan isitilah lain dari gambar, yang mana merupakan salah satu komponen multimedia yang memegang peranan penting sebagai bentuk informasi visual. Agar dapat diolah oleh komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Representasi fungsi molar (*continue*) menjadi nilai-nilai diskrit yang disebut dengan digitalisasi. Citra yang dihasilkan inilah yang disebut dengan istilah citra digital (*digital image*) (Munir, 2004).

Bagian terkecil yang menyusun citra digital disebut dengan piksel (*pixel*). Kedalaman warna yang direpresentasikan piksel bergantung pada jumlah bit per piksel yang dipakai. Berikut ini merupakan kedalaman warna yang biasa digunakan pada citra digital:

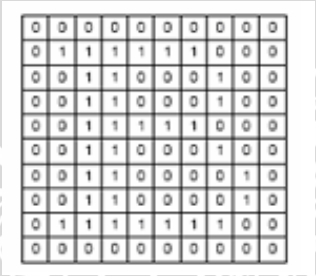
- 1-bit : *binary valued image* (21 = 2 warna)
- 8-bit : *grayscale* level (28 = 256 warna)
- 16-bit : *high color* (216 = 65.536 warna)
- 24-bit : *true color* (224 = 16.777.216 warna)
- 32-bit : *true color* (232 = 4.294.967.296 warna)

Berdasarkan warna-warna penyusunnya, citra digital dibagi menjadi tiga macam kategori yaitu (Burger dan Burge, 2007) :

1) Citra biner, yaitu citra yang hanya terdiri dari dua macam warna saja yakni hitam dan putih. Setiap piksel direpresentasikan dengan 1-bit. Gambar 2.9 merupakan gambar citra biner, dimana representasi citra binernya diperlihatkan pada gambar 2.10.



Gambar 2.9 Citra Biner



Gambar 2.10 Representasi Citra Biner

Citra biner hanya mempunyai dua nilai derajat keabuan yakni hitam dan putih. Piksel-piksel obyek bernilai 1 dan piksel-piksel latar belakang bernilai 0. Pada waktu menampilkan gambar, 0 adalah putih dan 1 adalah hitam.

2) Citra *grayscale*, yaitu citra dengan nilai piksel yang direpresentasikan dengan derajat keabuan atau intensitas warna putih. Intensitas warna paling rendah adalah warna hitam (0) sedangkan intensitas tingginya adalah warna putih (255). Pada umumnya *grayscale* memiliki kedalam warna 8- bit. Citra *grayscale* ditunjukkan pada gambar 2.11.

Gambar 2.11 Citra Grayscale

3) Citra berwarna, yaitu citra yang merepresentasikan warna tertentu. Banyak warna yang digunakan bergantung pada kedalam piksel yang bersangkutan. Citra berwarna direpresentasikan dalam beberapa kanal (*channel*) yang menyatakan komponen-komponen warna penyusunnya. Banyak kanal yang digunakan bergantung pada model warna yang digunakan citra tersebut. Intensitas suatu titik citra berwarna merupakan kombinasi dari tiga intensitas yakni derajat merah, hijau dan biru (RGB). Persepsi visual citra berwarna umumnya lebih kaya dibandingkan dengan citra hitam putih. Warna yang ditampilkan merupakan hasil kombinasi cahaya dengan panjang gelombang yang berbeda. Citra berwarna ditunjukkan pada gambar 2.12.

Gambar 2.12 Citra Berwarna

2.3.1 Bitmap

Citra digital bitmap secara teknis sering disebut dengan *raster images*. Citra digital jenis ini secara teknis tersusun atas titik-titik piksel. Citra bitmap banyak digunakan pada media elektronik berbasis pada warna yang kontinyu atau citra dengan gradasi warna, misalnya dalam dunia fotografi digital ataupun menggambar digital. Citra bitmap adalah citra digital yang *resolution-depent*, citra yang bergantung pada resolusi. Resolusi adalah jumlah piksel persatuan luas, misalnya 72 piksel/inci^2 yang merupakan ukuran standart dari citra bitmap. Sebuah citra bitmap tersusun atas piksel-piksel yang posisinya tertentu dengan nilai kedalam warna yang tertentu. Apabila resolusi piksel yang digunakan rendah, maka citra bitmap tampak bergerigi.

Citra dalam format bitmap lebih bagus daripada citra dengan format yang lain karena dalam citra bitmap umumnya informasi yang ada didalamnya tidak dimampatkan. Citra bitmap ini dibagi menjadi tiga macam yakni : citra biner, citra berwarna, dan citra hitam putih (*grayscale*).

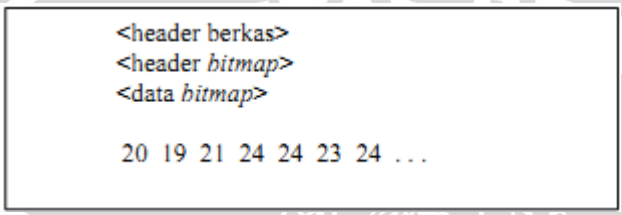
Header bitmap terdiri dari header berkas, header bitmap, informasi palet dan data bitmap. Format dari bitmap ditunjukkan pada gambar 2.13.

Header berkas	Header bitmap	Informasi palet	Data bitmap
14 byte	12 s/d 64 byte	0 s/d 1024 byte	N byte

Gambar 2.13 Format Berkas Bitmap 8-bit (Munir, 2004c)

Header adalah data yang letaknya pada bagian awal dari berkas. Data dalam *header* ini berguna untuk mengetahui bagaimana citra bitmap dikodekan dan disimpan. Letak informasi palet berada disebelah header bitmap. Informasi palet ini dinyatakan dengan tabel RGB. Data bitmap berada disebelah informasi palet. Data bitmap dalam berkas disusun secara terbalik dari bawah ke atas secara matrik pada ukuran *height x width*. Data piksel pada baris terbawah dinyatakan pada matirk bitmap baris ke-0. Sedangkan baris terakhir menyatakan data piksel baris teratas. Pada bagian data bitmap inilah pesan dapat disimpan dengan teknik steganografi LSB.

Citra 4-bit dan citra 8-bit menggunakan tabel informasi palet pada entri ke-*k*. Dimana *k* merupkan nilai dengan rentang 0-15 untuk citra 16 warna dan 0-255 untuk citra 256 warna. Berkas citra 24-bit tidak mempunyai palet RGB, karena nilai RGB langsung diuraikan dalam data bitmap. Setiap elemen data bitmap panjangnya 3 *byte*, masing-masing *byte* menyatakan *R*, *G*, dan *B*. Contoh format citra 24-bit seperti pada gambar 2.14.



Gambar 2.14 Format Citra 24-bit

2.4 Steganografi

Steganografi berasal dari bahasa Yunani yaitu *steganos* yang berarti menyembunyikan dan *graptos* yang artinya tulisan, secara keseluruhan artinya adalah tulisan yang disembunyikan. Secara umum steganografi merupakan seni atau ilmu yang digunakan untuk menyembunyikan pesan rahaisa dengan segala cara sehingga selain orang yang dituju, maka orang lain tidak akan menyadari keberadaan dari pesan rahasia tersebut (Munir, 2004).

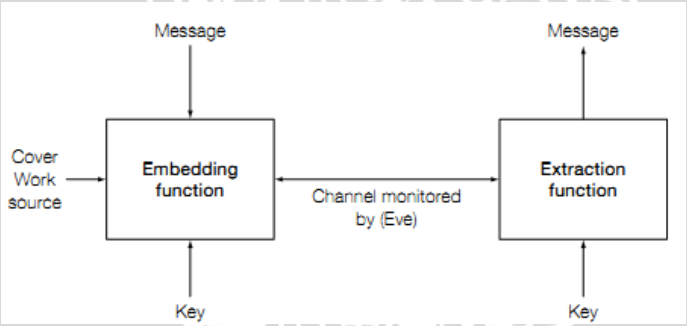
Terdapat dua proses utama dalam steganografi yaitu penyisipan (*embedding*) dan penguraian (*extraction*). *Embedding* merupakan proses menyisipkan pesan atau informasi ke dalam citra penampung (*cover image*), sedangkan *extraction* merupakan penguraian pesan yang tersembunyi di dalam citra stego. Pesan yang disembunyikan dalam sebuah citra membutuhkan dua buah properti. Pertama adalah citra asli yang belum dimodifikasi yang disebut dengan citra penampung. Properti yang kedua adalah informasi pesan yang akan disisipkan. Pesan ini bisa berupa plainteks, chiperteks, gambar lain, atau apapun yang dapat ditempelkan ke dalam bit *stream*. Ketika dikombinaksikan, citra penampung dan

pesan yang ditempelkan akan menghasilkan citra stego (*stego image*).

Kualitas dari steganografi dikatakan baik apabila memenuhi kriteria berikut ini (Munir, 2004c) :

1) *Fidelity*, penyisipan pada citra penampung tidak merubah mutunya. Setelah disisipkan pesan, maka tidak banyak perubahan yang terjadi, sehingga pengamat tidak sadar kalau di dalamnya terdapat pesan rahasia.
2) *Robustness*, informasi yang disisipkan di citra penampung mampu dipertahankan walaupun dilakukan operasi manipulasi pengolahan citra seperti pengubahan kontras, penajaman, kompresi, rotasi, pemotongan dan sebagainya.
   Recovery, Informasi yang disembunyikan harus dapat diungkapkan kembali ke bentuk awalnya.

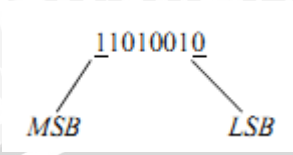
Proses steganografi secara umum ditunjukkan pada gambar 2.15.



Gambar 2.15 Proses Steganografi (Cox, dkk., 2007)

2.4.1 LSB (Least Significant Bit)

Penyisipan pesan dilakukan dengan cara mengganti bit-bit data di dalam segmen citra dengan bit-bit data rahasia. Metode pergantian bit yang paling sederhana adalah metode LSB (*Least Significant Bit*). Pada susunan bit di dalam 1 *byte*, terdapat bit yang paling berarti atau MSB (*Most Significant Bit*) dan bit yang kurang berarti atau LSB (*Least Significant Bit*). Pada gambar 2.16 ditunjukkan dimana letak MSB dan LSB.



Gambar 2.16 Letak MSB dan LSB (Munir, 2004c)

Bit yang paling cocok diganti adalah bit LSB, karena perubahan tersebut hanya mengubah nilai satu bit saja yang tidak berdampak terlalu besar jika dilakukan perubahan.

2.4.2 Penyisipan

Penyisipan LSB untuk berkas bitmap 24-bit dilakukan pada bit-bit terakhir dari nilai RGB-nya. Sebelum proses penyisipan, maka berkas yang akan dimasukkan harus dirubah terlebih dahulu ke bentuk biner. Misalnya terdapat susunan piksel bitmap sebagai berikut :

1010010	11001111	00110100
1011010	01001111	10110100
1110010	01101110	01100001

Sebagai contohnya huruf yang akan disisipkan adalah huruf ’s’, dimana representasi binernya adalah 01110011, maka akan didapatkan hasil :

1010010	11001111	00110101
1011011	01001110	10110100
1110011	01101111	01100001

Dari proses penyisipan diperoleh delapan bit terendah mengalami proses perubahan. Secara mata manusia perubahan bit ini tidak akan tampak perubahannya. Kenyataan inilah yang digunakan sebagai landasan teknik steganografi. Pesan yang dapat disembunyikan dengan metode steganografi LSB sangat terbatas pada ukuran dari citra penampung. Misalkan saja pada cita 24-bit yang berukuran 30x30 piksel terdapat 900 piksel. Setiap piksel

berukuran 3 *byte* dari komponen RGB. Berarti keseluruhan ada 900 x 3 = 2700 *byte*. Karena pada setiap byte hanya dapat menyembunyikan satu bit LSB, maka pesan yang dapat ditampung maksimal adalah 2700 / 8 = 337 *byte* atau 337 karakter. Namun ukuran ini harus dikurangi dengan panjang nama berkas, karena penyembunyian data rahasia tidak hanya menyembunyikan isi data tersebut tetapi juga nama berkasnya. Semakin besar data yang akan disembunyikan, maka semakin besar pula kemungkinan data tersebut akan rusak karena manipulasi pada citra penampung.

2.4.3 Penguraian

Pesan yang telah disembunyaikan pada citra stego dapat diungkap kembali dengan cara menguraikannya. Penguraian dilakukan dengan membaca tiap *byte* pada RGB yang menyimpan bit-bit pesan. Karena pada waktu proses penyisipan bersifat sekuensial, maka proses penguraiannya pun bersifat sekuensial. Indeks *byte* dibaca dari posisi piksel pojok kiri atas sampai pojok kanan bawah.

2.5 PSNR dan MSE

PSNR (*Peak Signal to Noise Ratio*) digunakan untuk mengukur perbandingan antara nilai maksimum dari sinyal yang diukur dengan besarnya derau yang berpengaruh pada sinyal tersebut. Dengan pengukuran PSNR dapat diketahui perbandingan kualitas citra sebelum dan sesudah disisipi pesan. Nilai PSNR menggunakan satuan *decibel* (db). Semakin besar nilai PSNR, maka semakin lebih baik kualitas citra tersebut. Rumus untuk PSNR ditunjukkan pada persamaan 2.7 (Sharma, dkk., 2011).

$$PSNR = 10\ log_{10}\left(\frac{255^2}{MSE}\right) \quad (2.7)$$

MSE (*Mean Square Error*) sendiri merupakan nilai rata-rata dari jumlah kuadat *absolute error* antara cita asli dengan citra stego. Persamaan MSE ditunjukkan pada persamaan 2.8

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{MN}} \sum_{\boldsymbol{Y=1}}^{\boldsymbol{M}} \sum_{\boldsymbol{X=1}}^{\boldsymbol{N}} \left(\boldsymbol{l}(\boldsymbol{x},\boldsymbol{y}) - \boldsymbol{l'}(\boldsymbol{x},\boldsymbol{y}) \right)^{\mathbf{2}} \quad (2.8)$$

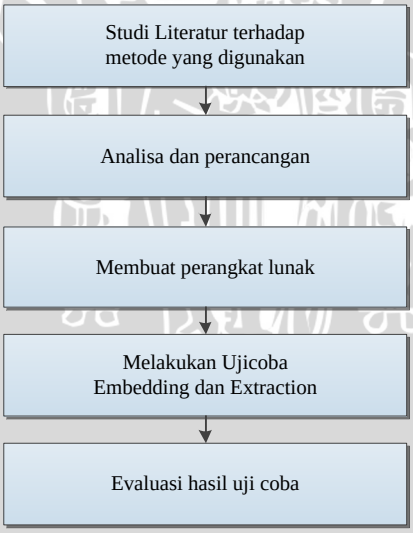
Keterangan :
M = panjang dari citra (ukuran piksel)
N = lebar dari citra (ukuran piksel)
(x,y) = koordinat masing-masing piksel
l = nilai bit pada citra asli
l' = nilai bit pada citra stego

BAB III
METODOLOGI DAN PERANCANGAN

Dalam bab metodologi dan perancangan ini, membahas tentang metode, rancangan, serta langkah-langkah yang harus dilakukan dalam pembuatan perangkat lunak ini. Langkah-langkah pembuatan perangkat lunak ini dijelaskan sebagai berikut :

1) Melakukan studi literatur terhadap sumber-sumber yang mendukung untuk pembuatan perangkat lunak ini, seperti yang dijelaskan pada bab 2.
2) Melakukan analisa dan merancang perangkat lunak dengan menerapkan metode yang digunakan.
3) Membuat perangkat lunak dari hasil analisis dan perancangan yang telah dilakukan.
4) Melakukan uji coba penyisipan dan penguraian dengan perangkat lunak yang telah dibuat.
5) Melakukan evaluasi hasil uji coba.

Gambar 3.1 merupakan alur dari proses pembuatan perangkat lunak tersebut.



Gambar 3.1 Diagram Alir Pembuatan Perangkat Lunak

3.1 Analisa Perangkat Lunak

3.1.1 Dekripsi Umum Perangkat Lunak

Perangkat lunak yang dikembangkan ini digunakan untuk melakukan penyisipan dan penguraian pesan pada citra bitmap 24-bit. Metode yang digunakan pada dasarnya menggunakan steganografi namun untuk meningkatkan tingkat keamanannya dan memperbesar jumlah ukuran pesan yang ingin disisipkan maka dilakukan beberapa modifikasi pada steganografi ini. Dengan mengkombinasikan kriptografi DES, kompresi Huffman dan steganografi LSB diharapkan mampu menghasilkan kualitas citra stego yang memenuhi kriteria tersebut.

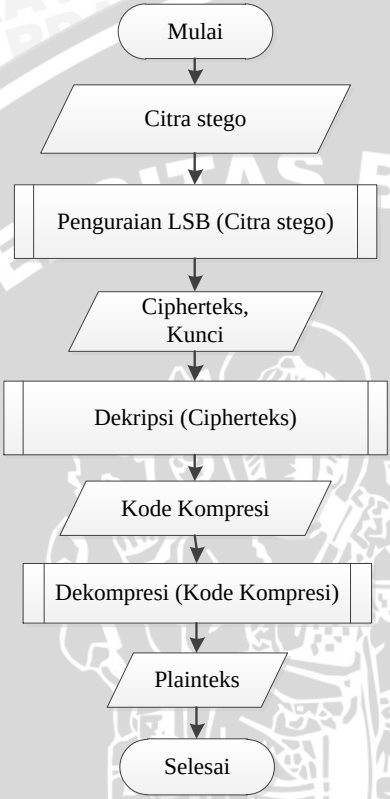
Masukkan pada perangkat lunak ini ada tiga macam yakni citra bitmap dengan kedalaman piksel 24-bit, pesan yang akan disisipkan serta kunci untuk mengamankan pesannya. Dalam proses penyisipan, terlebih dahulu pesan akan dikompres dengan Huffman. Kompresi Huffman yang bersifat *lossless* menjamin isi dari informasi yang dikompres tidak ada yang hilang. Dari proses kompresi Huffman dihasilkan tabel Huffman dan serangkaian kode Huffman. Untuk mengamankan kode Huffman tersebut, maka dilakukan enkripsi DES. Enkripsi DES bekerja dengan masukkan kunci 8 karakater yang akan dijadikan kunci internal. Dari kunci internal inilah akan dilakukan enkripsi terhadap kode Huffman. Hasil dari enkripsi akan menghasilkan cipherteks. Cipherteks selanjutnya akan disisipkan ke citra penampung. Proses penyisipannya menggunakan metode steganografi LSB. Metode LSB mengganti bit-bit terakhir RGB per piksel dari citra penampung dengan cipherteks. Serangkaian dari proses tersebut akan menghasilkan citra stego yang di dalamnya terdapat pesan rahasia. Pada gambar 3.2 ditunjukkan proses dari penyisipan pesan dalam perangkat lunak ini.

Gambar 3.2 Penyisipan pesan

Supaya pesan dapat dibaca kembali maka diperlukan proses penguraian pada citra stego. Terdapat dua parameter masukkan yang diperlukan dalam proses penguraian yakni citra stego dan kunci. Pertamakali citra stego akan diproses dengan penguraian LSB. Dengan mengambil tiap bit terakhir pada setiap *byte* RGB per piksel maka diperoleh cipherteks.

DES akan mendekripsi cipherteks tersebut dengan parameter kunci yang telah dimasukkan. Jika kunci yang dimasukkan sama dengan kunci pada proses enkripsi, maka diperoleh kode Huffman yang benar. Namun beda halnya jika kunci yang dimasukkan tidak sama dengan proses enkripsi, hasil yang diperoleh adalah kode Huffman yang tidak salah. Proses berikutnya, kode Huffman akan di dekompresi berdasarkan tabel Huffman. Penelusuran melalui pohon Huffman pada proses dekompresi akan menghasilkan kembali pesan

rahasianya. Pada gambar 3.3 ditunjukkan proses dari penguraian pesan dalam perangkat lunak ini.



Gambar 3.3 Penguraian Pesan

3.1.2 Batasan Perangkat Lunak

1. Media yang digunakan untuk citra penampung adalah citra bitmap (*.bmp) 24-bit minimal 3x8 piksel atau 8x3 piksel.
2. Pesan rahasia yang disisipkan berupa berkas teks (*.txt) atau *string*.
3. Kunci yang digunakan berupa *string* dengan panjang maksimal 8 karakter atau 64-bit.

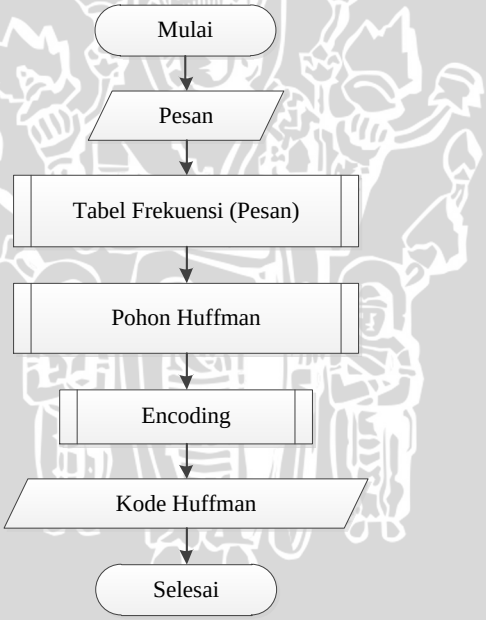
4. Keluaran yang dihasilkan berupa citra stego dengan format yang sama dengan citra penampungnya (*.bmp).

3.2 Perancangan Perangkat Lunak

3.2.1 Perancangan Proses Penyisipan

Dalam sub bab berikut dijelaskan tentang perancangan perangkat lunak ini. Proses pertama adalah mengkompres pesan menjadi kode Huffman. Dari proses kompresi dilanjutkan ke enkripsi DES. Hasil enkripsi DES yang berupa cipherteks disisipkan ke citra penampung dengan metode LSB untuk mendapatkan citra stego.

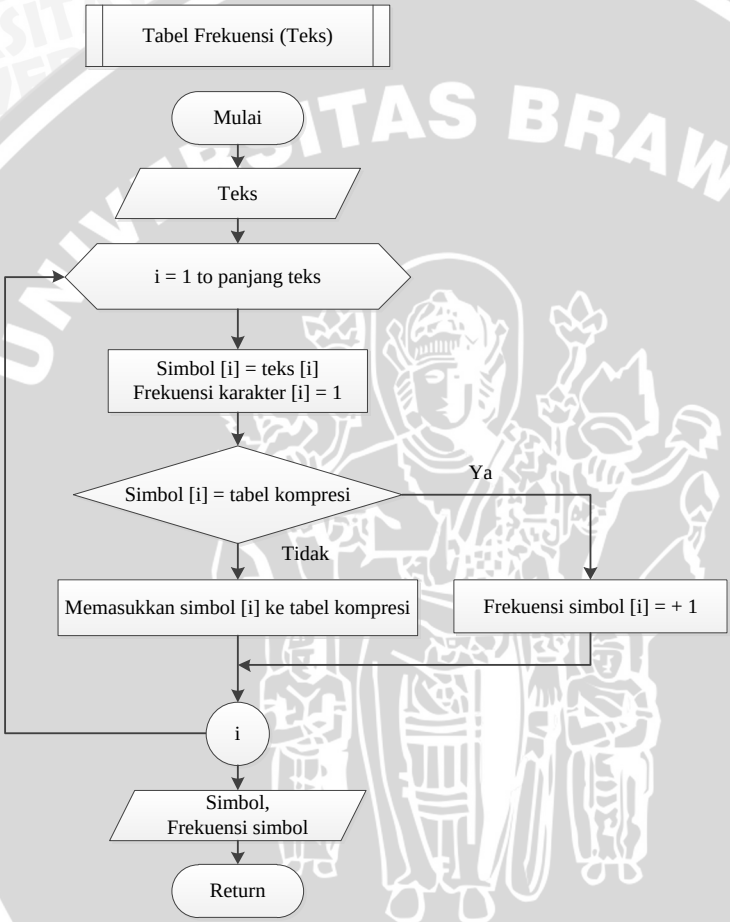
A. Kompresi Huffman



Gambar 3.4 Diagram Alir Kompresi Huffman

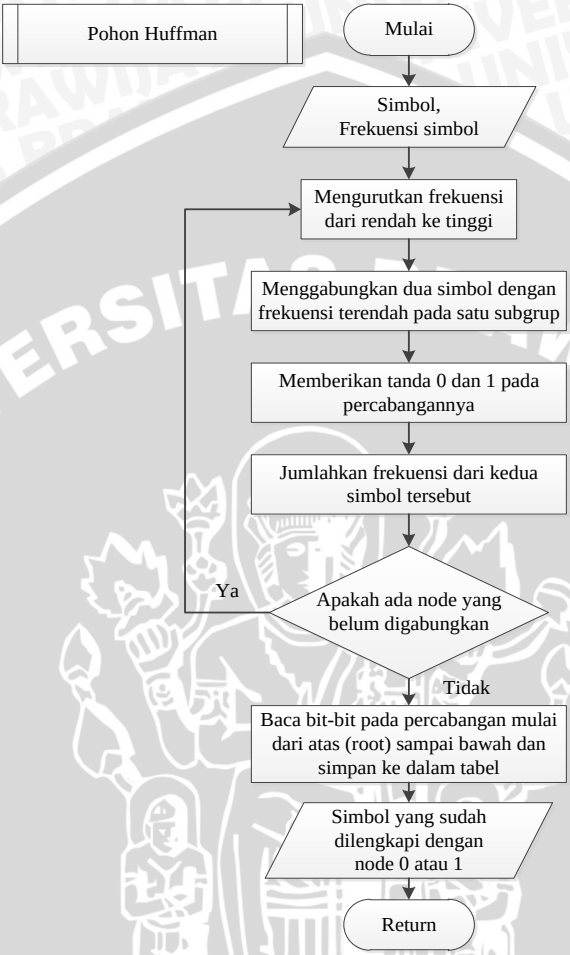
Berdasarkan gambar 3.4 maka langkah pertama yang dilakukan untuk melakukan kompresi Huffman adalah membuat tabel frekuensi. Tabel Frekuensi ini dibentuk sebagai acuan untuk

enkoding data yang akan dikompresi. Setiap simbol dari pesan dimasukkan ke tabel berdasarkan tingkat frekuensi kemunculannnya. Diagram alir dari proses pembuatan tabel frekuensi ditunjukkan pada gambar 3.5.



Gambar 3.5 Diagram Alir Pembentukan Tabel Frekuensi

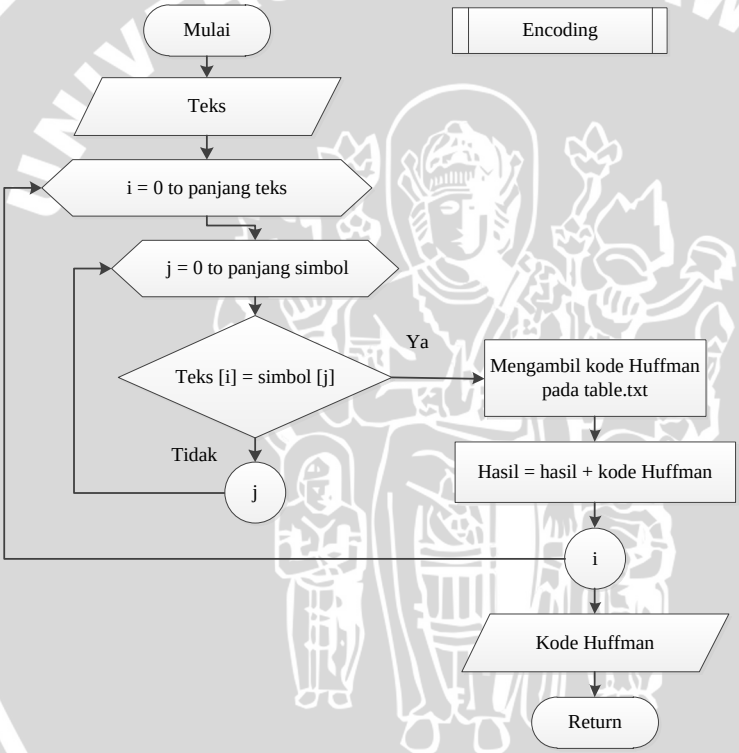
Setelah terbentuk tabel frekuensi, tahap selanjutnya adalah membentuk pohon Huffman-nya. Pada gambar 3.6 ditunjukkan diagram alir dari proses pembentukan pohon Huffman.



Gambar 3.6 Diagram Alir Pembentukan Pohon Huffman

Pembentukan pohon Huffman dimulai dengan mengurutkan simbol sesuai frekuensinya secara *ascending* (rendah ke tinggi). Berikutnya kedua simbol dengan frekuensi terkecil digabung menjadi satu membentuk sebuah akar baru. *Node* sebelah kiri diberi label ‘0’ dan *node* sebelah kanan diberi label ‘1’. Penggabungan simbol tersebut akan membuat sebuah akar baru dimana frekuensinya adalah jumlah dari kedua simbol sebelumnya. Bandingkan kembali frekuensi akar tersebut dengan frekuensi simbol

yang lain pada tabel kompresi. Proses tersebut diulangi sampai semua simbol diakses semua sehingga terbentuk satu pohon Huffman. Hasil dari pembentukan pohon ini disimpan ke dalam suatu berkas bernama Table.txt yang meyimpan kode simbol dan kode Huffman. Langkah akhir adalah proses enkoding (*encoding*) untuk menyusun pesan berdasarkan kode Huffman yang telah terbentuk. Proses enkoding dilakukan dengan cara menelusuri tiap *node* pada pohonnya. Diagram alir dari proses enkoding ditunjukkan pada gambar 3.7.



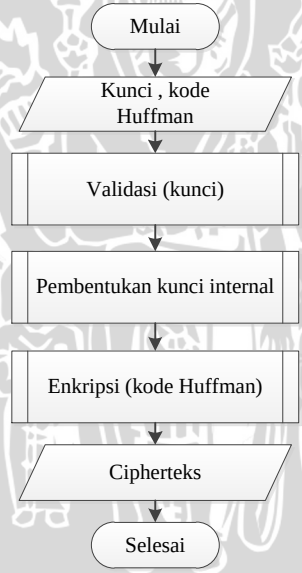
Gambar 3.7 Diagram Alir Enkoding Kode Huffman

Berdasarkan gambar 3.7, langkah yang harus dilakukan pertama kali adalah menentukan bagian pesan mana yang akan di enkoding terlebih dahulu. Pembacaan dimulai dari akar dengan cara membaca setiap label dari cabang yang bersesuaian sampai bertemu

daun (*leaf*) dimana simbol dari pesan tersebut berada. Hal tersebut diulangi sampai seluruh pesan selesai di enkoding.

B. Enkripsi DES

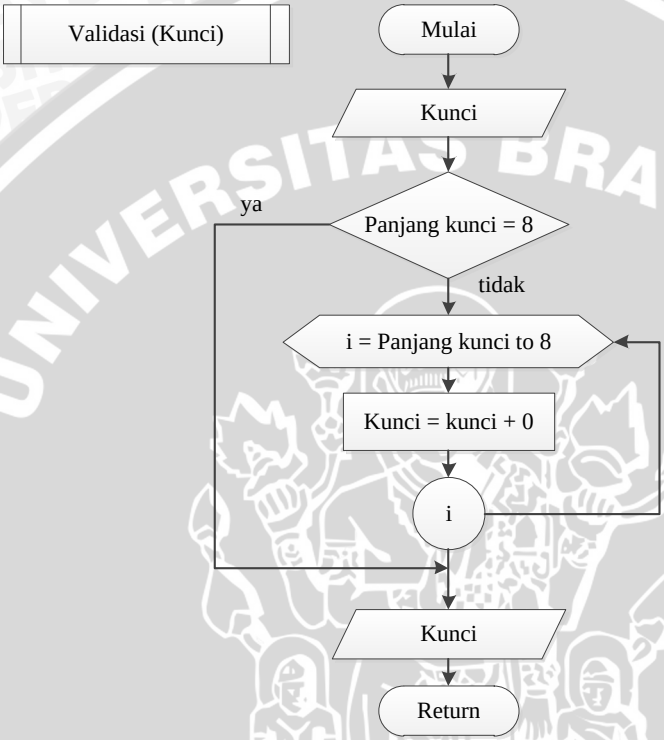
Proses enkripsi DES diawali dengan memasukkan kunci serta kode Huffman yang nantinya ingin disisipkan ke citra penampung. Untuk memastikan apakah kunci yang dimasukkan sudah valid maka dilakukan proses pengecekan. Jika valid, tahap selanjutnya adalah membentuk kunci internalnya. Kunci internal ini diperoleh melalui beberapa tahap yang dijelaskan pada sub bab 2.1.1. Terbentuknya 16 kunci internal ini akan di proses lebih lanjut untuk mengenkripsi kode Huffman. Diagram alir dari proses enkripsi DES secara keseluruhan ditunjukkan pada gambar 3.8.



Gambar 3.8 Diagram Alir Enkripsi DES

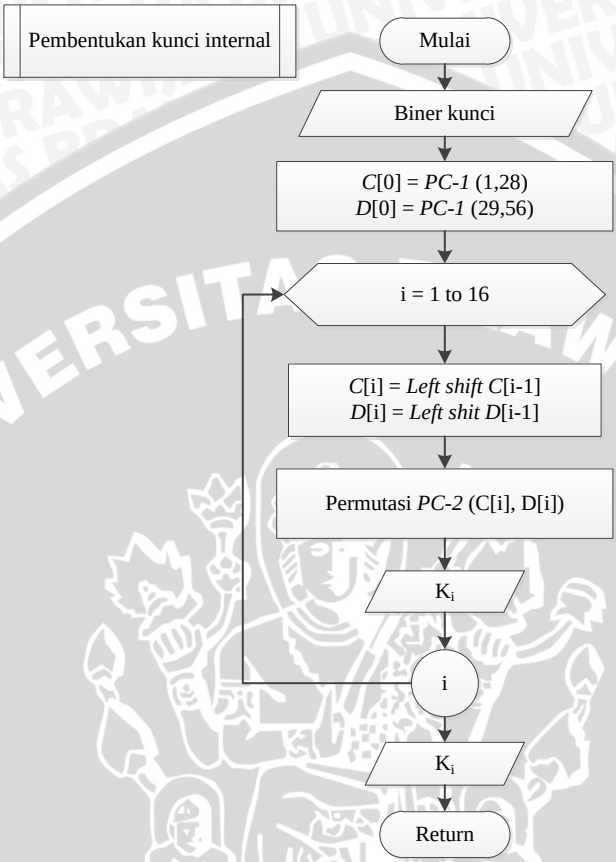
DES bekerja pada kunci eksternal dengan panjang 8 karakter atau 64-bit. Diperlukan suatu proses validasi kunci dimana untuk memastikan panjangnya. Pada validasi kunci, apabila jumlah panjang dari kunci lebih dari 8 karakter, maka return akan bernilai null.

Namun jika kurang dari 8 karakter, akan dilakukan penambahan karakter berupa angka ‘0’ sampai panjang kuncinya menjadi 8 karakter. Proses dari validasi kunci ditunjukkan pada gambar 3.9.



Gambar 3.9 Diagram Alir Validasi Kunci

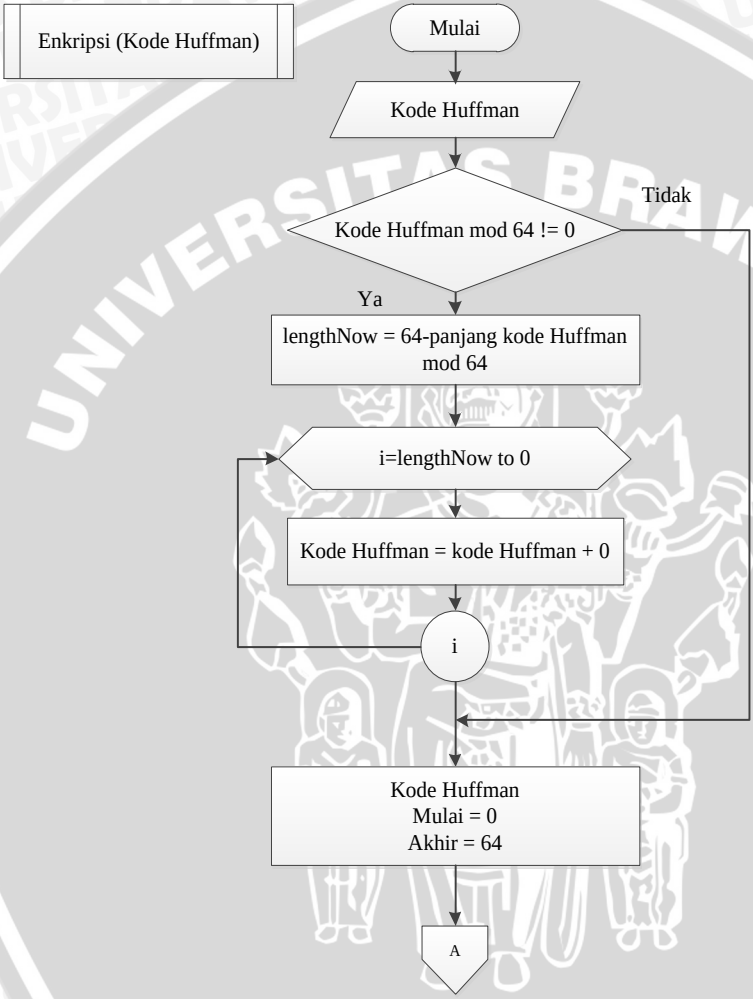
Ketika kunci sudah valid dengan jumlah 8 karakter, maka tahapan selanjutnya adalah melakukan pembentukan kunci internalnya. Kunci terlebih dahulu dirubah ke dalam bentuk biner. Biner kunci yang terbentuk berikutnya dipermutasi dengan *PC-1* sesuai tabel 2.1. Hasil permutasinya dibagi menjadi dua bagian C_0 dan D_0. Tahap berikutnya adalah melakukan penggeseran bit-bit C_i dan D_i ke kiri sesuai tabel 2.2. Hasil dari pergeseran tiap bit-bit tersebut selanjutnya dipermutasi dengan *PC-2* berdasarkan tabel 2.3 selama 16 putaran untuk mendapatkan kunci internal K_i. Proses dari pembentukan kunci internal ditunjukkan pada gambar 3.10.

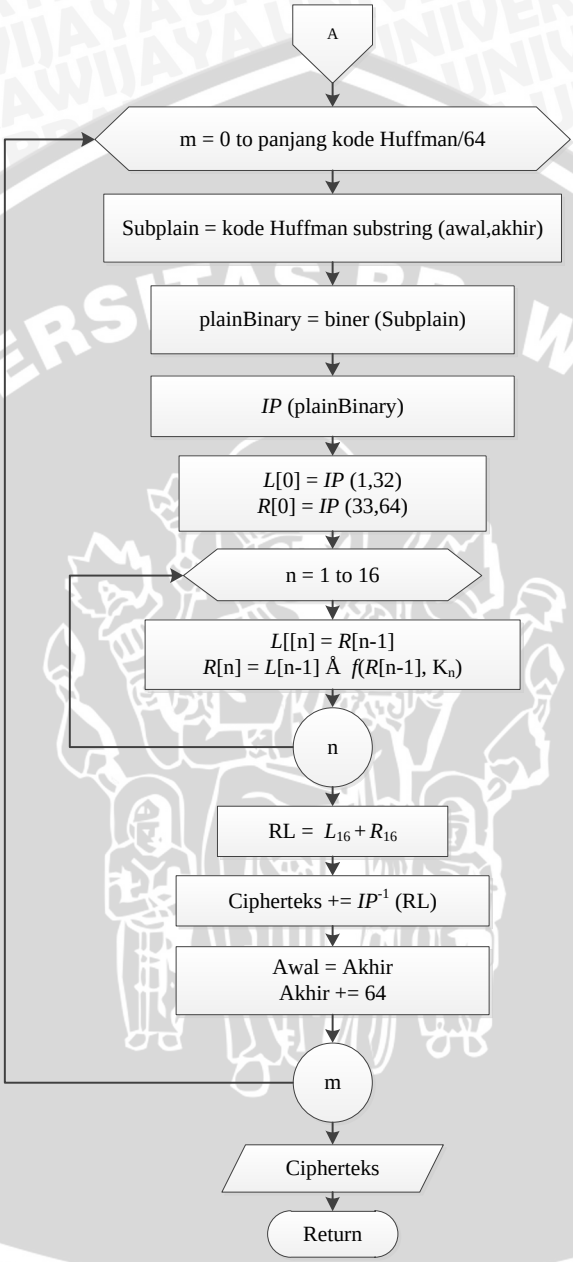


Gambar 3.10 Diagram Alir Pembentukan Kunci Internal

Proses enkripsi kode Huffman menjadi cipherteks dilakukan setelah kunci internal tersebut dibentuk. Panjang kode Huffman di cek terlebih dahulu apakah kelipatan 64 atau tidak. Apabila kurang dari kelipatan 64 maka harus ditambahi bit '0' sampai jumlahnya menjadi kelipatan 64. Dari setiap 64-bit blok kode huffman tersebut dilewatkan melalui matrik permutasi awal (*IP*) sesuai tabel 2.4. Hasil dari permutasi tersebut kemudian dibagi menjadi 2 bagian yakni 32-bit bagian R_0 dan 32-bit bagian L_0. R_0 dan L_0 mengalami 16 kali putaran, dimana setiap putarannya merupakan jaringan Feistel yang dijelaskan pada persamaan 2.2. Keluaran dari jaringan Feistel

dipermutasi dengan matrik permutasi invers (IP^{-1}) untuk memperoleh cipherteks. Proses enkripsi DES ditunjukkan pada gambar 3.11.



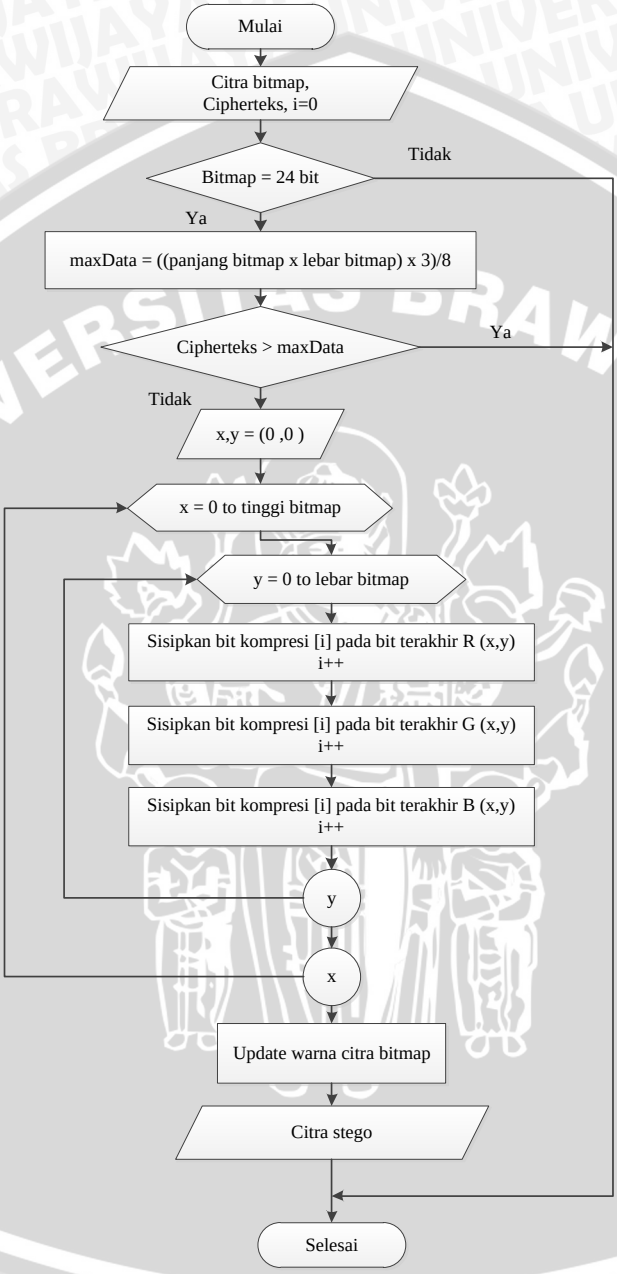


Gambar 3.11 Diagram Alir Enkripsi Kode Huffman

C. Penyisipan LSB

Proses penyisipan menggunakan steganografi LSB dengan cara menyisipkan bit-bit cipherteks yang sudah dikompresi ke dalam bit-bit citra penampung. Langkah-langkah dalam penyisipan pesan dengan steganografi LSB sebagai berikut :

1. Terdapat dua parameter untuk masukkan, yakni citra penampung berformat bitmap dan bit pesan yang ingin disisipkan.
2. Melakukan pengecekan terhadap citra penampung bitmap, apakah citra tersebut memiliki kedalam piksel 24- bit. Apabila kondisi terpenuhi, maka akan dilanjutkkan dengan proses selanjutnya. Namun apabila kondisi tidak terpenuhi, proses penyisipan langsung diakhiri.
3. Dilakukan proses pengecekan, apakah panjang bit pesan yang akan disisipkan melebihi kapasitas dari citra penampung. Apabila kondisi tersebut benar, maka proses akan dihentikan, namun jika kondisinya salah maka akan dilakukan proses perulangan untuk melakukan penyisipan.
4. Proses penyisipan dilakukan selama panjang dari bit pesan terpenuhi. Dilakukan perulangan untuk variabel x sampai batas tinggi dari citra penampung. Sedangkan variabel y diulang sampai batas lebar citra penampung.
5. Berikutnya dilakukan proses penyisipan berdasarkan piksel (x,y) pada RGB.
6. Langkah 4-5 diulangi sampai batas dari panjang pesan yang dimasukkan. Setelah selesai, citra penampung diperbaharui berdasarkan perubahan bit-bit yang sudah dilakukan perubahan sehingga menghasilkan citra stego. Gambar 3.12 menunjukkan diagram alir dari proses penyisipan LSB berdasarkan langkah-langkah 1 sampai 6.



Gambar 3.12 Diagram Alir Penyisipan LSB

Untuk mempermudah proses penguraian pada perangkat lunak ini maka perlu dilakukan penyimpanan panjang kode Huffman dan panjang cipherteks. Proses penyimpanan ini tidak dilakukan di dalam citra penampung supaya lebih efisien. Penyimpanan dilakukan diberkas eksternal yang dinamai SecretKey.ser.

3.2.2 Perancangan Proses Penguraian

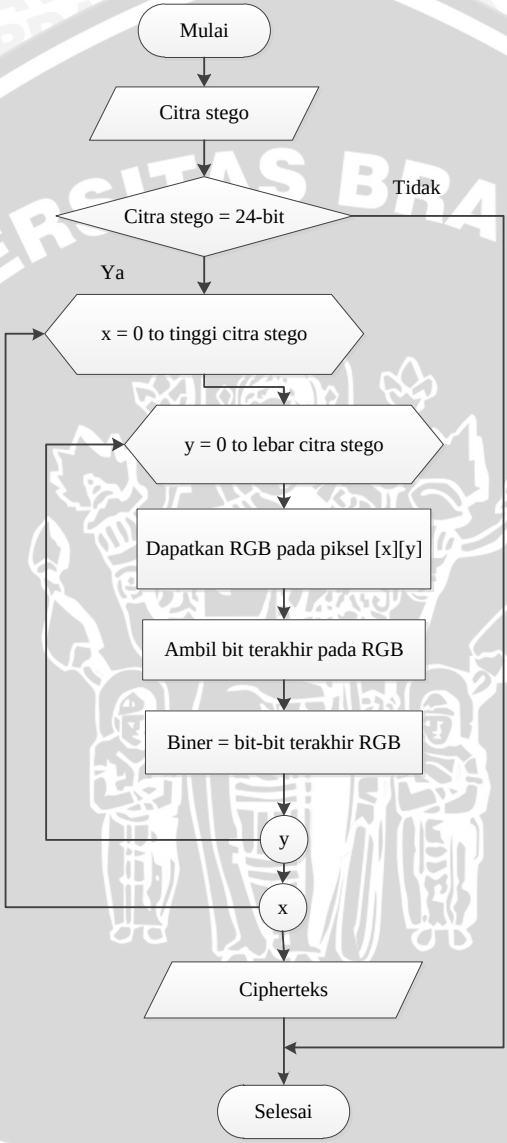
Dalam sub bab ini, dibahas mengenai proses penguraian pesan. Telebih dahulu pesan diekstraksi dengan LSB untuk mendapatkan cipherteks. Cipherteks ini selanjutnya di dekripsi untuk menghasilkan kode Huffman. Supaya kode Huffman dapat dibaca kembali menjadi informasi yang utuh, perlu sebuah metode dekompresi.

A. Ekstraksi LSB

Hasil citra stego yang menyimpan pesan rahasia yang harus diekstrak kembali untuk dapat membacanya. Langkah-langkah proses ekstraksi secara detail sebagai berikut :

1. Memasukkan citra stego yang berisi pesan rahasia.
2. Dilakukan pengecekan terhadap kedalaman piksel, apakah 24-bit atau tidak. Jika kondisi benar, maka dilanjutkan proses selanjutnya. Namun jika kondisi salah, proses akan dihentikan.
3. Penelusuran piksel dilakukan mulai dari batas lebar sampai batas tinggi dari citra.
4. Dalam satu piksel diperoleh nilai tiga komponen warna yakni RGB. Dari tiap RGB tersebut, diambil bit-bit terakhirnya saja kemudian disimpan ke dalam variabel biner.
5. Langkah 3-4 dilakukan perulangan sampai batas tinggi dan lebar dari citra stego.
6. Dari langkah 1-5 maka akan diperoleh hasil ekstraksi LSB berupa biner terkompresi.

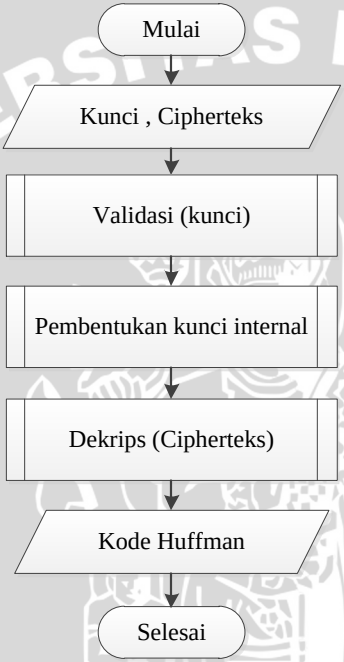
Pada gambar 3.13 ditunjukkan diagram alir dari proses ektraksi dengan metode LSB.



Gambar 3.13 Diagram Alir Ekstraksi LSB

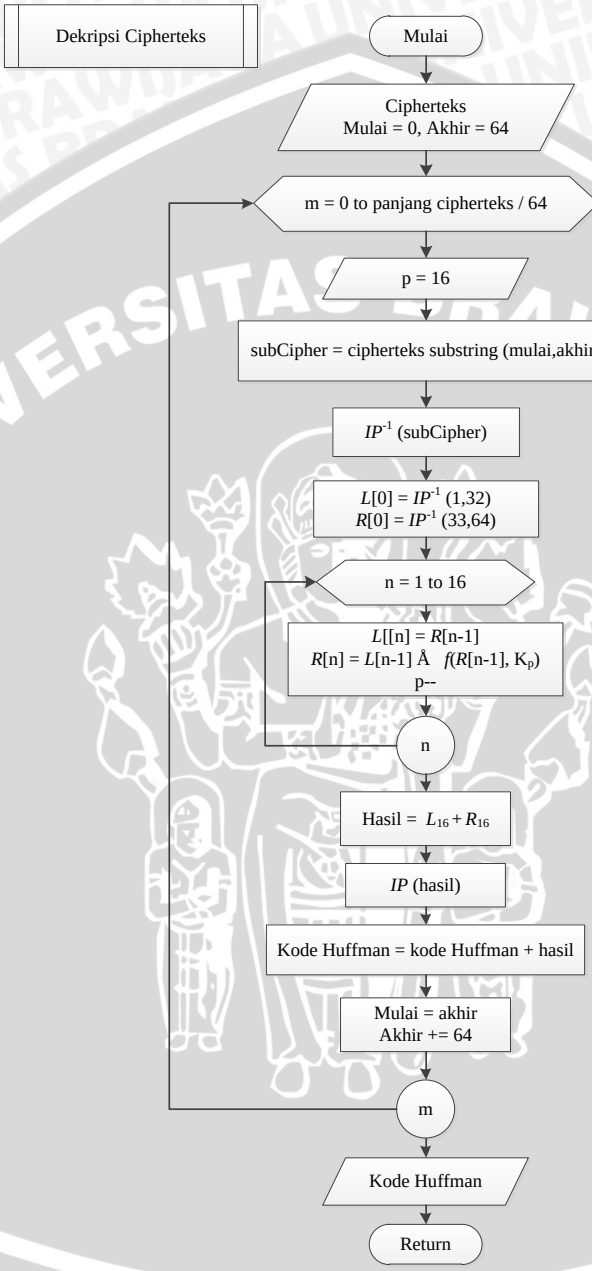
B. Dekripsi DES

Sesuai dengan ciri algoritma DES, yakni memiliki kunci simetri. Kunci simetri artinya kunci yang digunakan untuk enkripsi dan dekripsi adalah sama. Diagram alir dari proses dekripsi DES ditunjukkan pada gambar 3.14.



Gambar 3.14 Diagram Alir Dekripsi DES

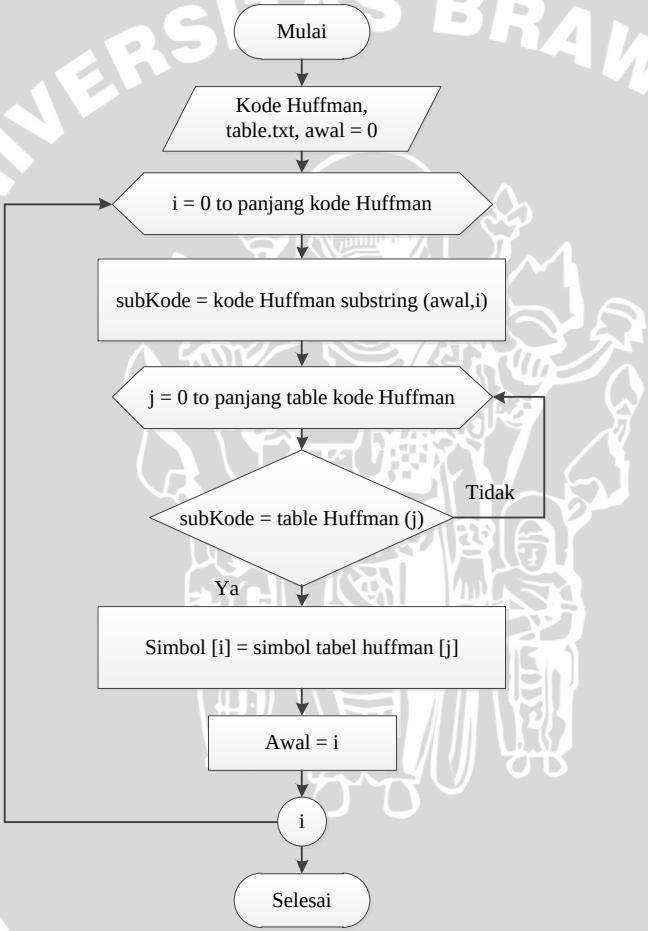
Proses validasi kunci dan pembentukan kunci internal sama dengan proses enkripsi. Pada proses dekripsi, semua proses hampir sama dengan enkripsi. Hal yang membedakannya adalah penggunaan kunci intenalnya. Pada proses enkripsi, kunci internal dimulai dari K_1, K_2, K_3, ... K_{16} maka dalam proses dekripsi kunci internalnya dibalik menjadi K_{16}, K_{15}, K_{14}, ... K_1. Dari proses dekripsi ini akan dihasilkan kembali plainteks sehingga pesan rahasia dapat dibaca. Untuk proses dekripsi cipherteks ditunjukkan pada gambar 3.15.



Gambar 3.15 Diagram Alir Dekripsi Cipherteks

C. Dekompresi Huffman

Proses dekompresi adalah menyusun kembali kode Huffman pada berkas terkompresi untuk mendapatkan kembali teks yang utuh. Pada aplikasi ini, untuk dekompresi digunakan metode pembacaan tabel kode Huffman. Gambar 3.16 menjelaskan diagram alir dari proses dekompresi Huffman.

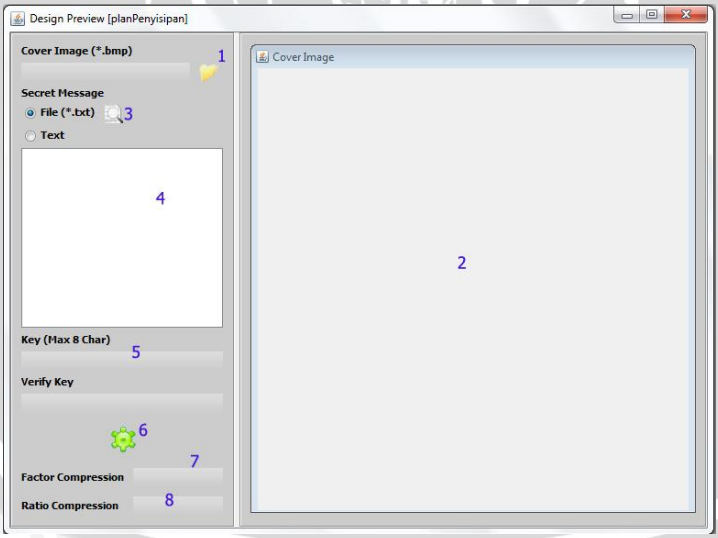


Gambar 3.16 Diagram Alir Dekompresi Huffman

Tabel kode Huffman sebelumnya telah dibuat dan disimpan ke dalam table.txt pada saat proses kompresi. Kode Huffman akan di dekompresi mulai indek yang pertama bagian kiri, lalu dicocokkan dengan tabel kode Huffman. Jika cocok maka ditulis ke variabel keluaran berupa simbol yang disimpan pada tabel Huffman. Namun jika tidak cocok, indek dari kode Huffman ditambah satu, sehingga ada dua kode Huffman. Kode tersebut kemudian dicocokkan kembali dengan tabel kode biner Huffman. Proses pencocokan pada tabel Huffman berhenti jika semua kode Huffman sudah di dekompresi semua. Dari sederetan proses tersebut akan dihasilkan kembali pesan rahasianya.

3.3 Perancangan Antarmuka

Desain antarmuka yang dibuat dibagi menjadi tiga bagian yakni bagian penyisipan, penguraian dan analisa. Untuk menyisipkan pesan ke dalam citra penampung, *user* hanya perlu memasukkan beberapa properti saja seperti citra penampung, berkas pesan yang akan disisipkan serta kunci untuk mengamankan pesan. Pada gambar 3.17 ditunjukkan gambar dari panel penyisipan.

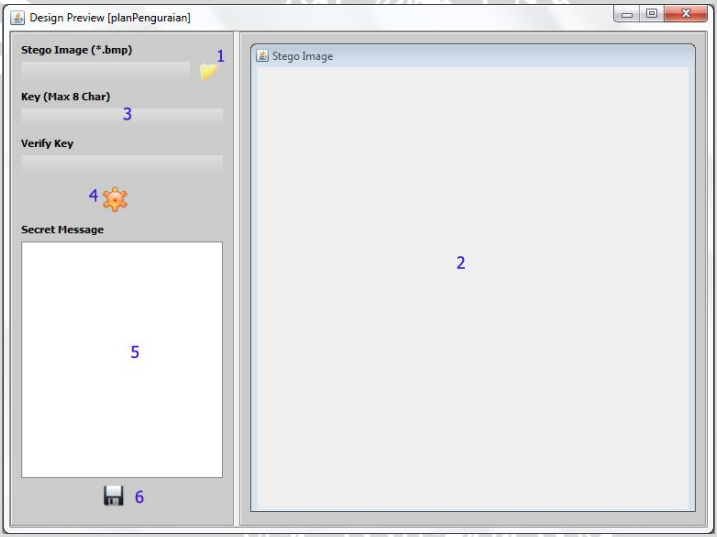


Gambar 3.17 Panel Penyisipan

Keterangan gambar 3.17 sebagai berikut :

1. Tombol untuk membuka citra penampung.
2. Pada *internal frame* ini citra penampung akan dimuat.
3. Pilihan untuk memilih pesan berupa berkas *.txt atau *string*.
4. Isi dari pesan akan ditampilkan di *text area*.
5. Kunci untuk enkripsi DES.
6. Tombol penyisipan.
7. Nilai dari faktor kompresi terhadap pesan yang di kompres.
8. Nilai dari rasio kompresi terhadap pesan yang di kompres.

Untuk melakukan penguraian *user* hanya memerlukan dua properti saja yakni citra stego dan kunci untuk dekripsi DES. Panel dari penguraian ditunjukkan pada gambar 3.18.



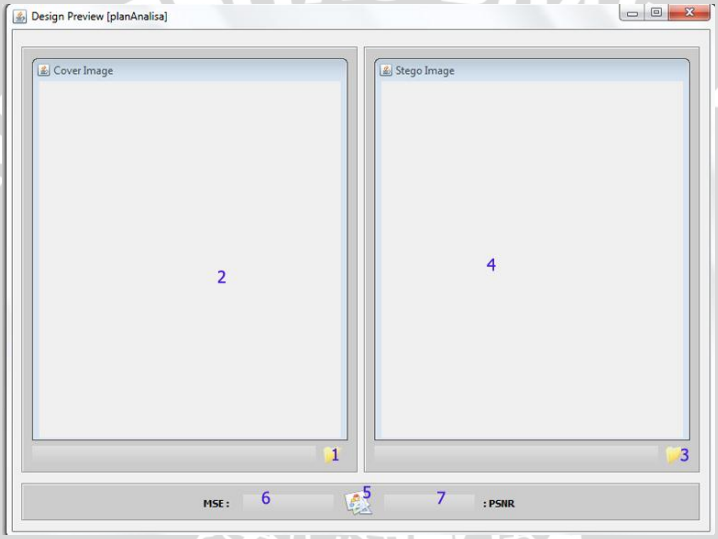
Gambar 3.18 Panel Penguraian

Keterangan gambar 3.18 :

1. Tombol untuk membuka citra stego.
2. Pada *internal frame* ini citra stego akan dimuat.
3. Kunci untuk dekripsi DES.
4. Tombol penguraian.

5. *Text area* yang akan menampilkan isi dari proses penguraian.
6. Tombol simpan yang digunakan untuk menyimpan hasil dari proses penguraian berupa berkas berformat *.txt.

Selain penyisipan dan penguraian, disediakan juga panel analisa sehingga *user* dapat mengetahui seberapa besar nilai MSE dan PSNR. Panel analisa ditunjukkan pada gambar 3.19.



Gambar 3.19 Panel Analisa

Keterangan gambar 3.19 :

1. Tombol untuk membuka citra penampung.
2. *Internal frame* yang memuat citra penampung.
3. Tombol untuk membuka citra stego.
4. *Internal frame* yang memuat citra stego.
5. Tombol analisa.
6. *Text field* yang menampilkan hasil dari MSE.
7. *Text field* yang menampilkan hasil dari PSNR.

3.4 Perancangan Uji Coba dan Evaluasi Hasil

Pada sub bab ini akan dibahas mengenai perancangan pengujian yang dilakukan terhadap perangkat lunak yang akan dibuat. Pengujian terhadap perangkat lunk ini dibagi menjadi tiga macam yakni pengujian fungsionalitas, kinerja, serta pengujian ketahanan citra stego yang dihasilkan.

3.4.1 Bahan Pengujian

Perangkat lunak yang dikembangkan memerlukan masukkan berupa citra penampung berformat bitmap dengan kedalaman piksel 24-bit. Selain masukkan citra bitmap, diperlukan juga masukkan berupa pesan yang akan disisipkan. Pesan yang disisipkan ini bisa berupa *string* atau berkas *.txt. Diperlukan juga masukkan kunci untuk enkripsi dan dekripsi pesan tersebut, agar tidak mudah dibaca.

3.4.2 Pengujian Fungsionalitas Perangkat Lunak

Pengujian fungsionalitas ini dibagi menjadi tiga bagian. Pertama pengujian terhadap kode Huffman. Pengujian pertama ini digunakan untuk mengetahui apakah berkas yang dikompres mampu didekompres kembali untuk mendapatkan plainteks yang utuh. Tabel 3.1 menunjukkan pengujian fungsionalitas terhadap kode Huffman.

Table 3.1 Pengujian Fungsionalitas Kode Huffman

Proses Kode Huffman			
No.	Nama Berkas	Kompresi	Dekompresi

Pengujian fungsionalitas kedua dilakukan terhadap proses kriptografi DES. Pengujian ini dilakukan untuk mengetahui apakah pesan yang sudah dienkripsi dapat dibaca kembali dengan proses dekripsi. Tabel 3.2 menunjukkan pengujian fungsionalitas terhadap kriptografi DES.

Table 3.2 Pengujian Fungsionalitas Kriptografi DES

Proses Kriptografi				
No.	Kode Huffman	Kunci	Enkripsi	Dekripsi

Untuk pengujian fungsionalitas ketiga dilakukan terhadap proses steganografi LSB. Pengujian fungsionalitasnya ditunjukkan pada tabel 3.3.

Table 3.3 Pengujian Fungsionalitas Steganografi LSB

Proses Steganografi				
No.	Nama Citra	Cipherteks	Penyisipan	Penguraian

3.4.3 Pengujian Kirnerja Perangkat Lunak

Pengujian kinerja perangkat lunak dibagi menjadi empat bagian yakni pengujian kinerja terhadap proses kode Huffman, kriptografi DES, steganografi LSB, serta waktu eksekusi. Pada pengujian kinerja kode Huffman dilakukan proses perhitungan rasio dan faktor kompresinya untuk mengetahui berapa persenkah pesan mampu dikompres.Pengujian kinerja kode Huffman ditunjukkan pada tabel 3.4.

Table 3.4 Pengujian Kinerja Kode Huffman

Proses Kode Huffman					
No.	Nama Berkas	Faktor	Rasio %	Kompresi	Dekompresi

Pengujian kinerja kriptografi DES yaitu dengan cara memasukkan kode Huffman dan dilakukan proses enkripsi untuk menghasilkan cipherteks. Dari cipherteks yang didapat dilakukan proses dekripsi untuk mendapatkan kembali kode Huffman-nya. Tabel 3.5 menunjukkan pengujian kinerja terhdap kriptografi DES.

Table 3.5 Pengujian Kinerja Kriptografi DES

Proses Kriptografi				
No.	Nama Berkas	Kunci	Enkripsi	Dekripsi

Pengujian kinerja selanjutnya dilakukan pada steganografi LSB. Perhitungan pada MSE dan PSNR digunakan untuk mengetahui hasil dari kualitas citra stego yang dihasilkan. Tabel 3.6 menunjukkan pengujian kinerja terhadap steganografi LSB.

Table 3.6 Pengujian Kinerja Steganografi LSB

Proses Steganografi					
No.	Nama Citra	Nama Berkas	Citra Stego	MSE	PSNR

Pada tabel 3.7 ditunjukkan rancangan pengujian untuk mengetahui sampai seberapa besar berkas pesan dapat ditampung oleh citra penampung.

Table 3.7 Pengujian Batas Maksimal Pesan Yang Disisipkan

No.	Nama Berkas	Panjang Bit Sebelum Dikompres	Panjang Bit Setelah Dikompres

Faktor waktu eksekusi dari perangkat lunak yang dikembangkan akan menjadi penentu apakah kombinasi algoritma yang digunakan mampu mengeksekusi proses penyisipan dan penguraian secara baik. Pengujian kinerja waktu eksekusi ditunjukkan pada tabel 3.8.

Table 3.8 Pengujian Kinerja Waktu Eksekusi

Proses Steganografi			
No.	Nama Citra	Nama Berkas	Waktu Eksekusi (detik)

3.4.4 Pengujian Ketahanan Citra Steganografi

Untuk menguji seberapa kuat ketahanan citra stego yang dihasilkan perlu dilakukan beberapa pengujian manipulasi citra. Manipulasi citra yang dilakukan meliputi rotasi, *cropping*, *noise*, serta *resize*. Pada tabel 3.9 ditunjukkan tabel pengujian terhadap citra stego yang dihasilkan.

Table 3.9 Pengujian Ketahanan Citra Stego

Manipulasi	Citra manipulasi	Hasil penguraian	Hasil
Rotation			
Cropping			
Noise			
Resize			

3.5 Perhitungan Manual

Pada sub bab ini membahas tentang perhitungan manual dari perangkat lunak yang dikembangkan. Perhitungan dibagi menjadi dua bagian yakni proses penyisipan dan proses penguraian. Sebagai contoh, pesan yang ingin disisipkan adalah ‘hadiah’ sedangkan kuncinya adalah ‘sync458r’.

3.5.1 Perhitungan Manual Pada Penyisipan

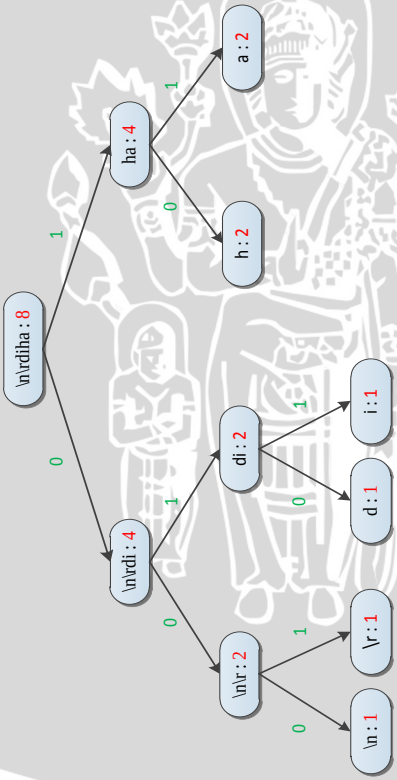
A. Kompresi Huffman

Pada kompresi Huffman, setiap simbol dihitung berdasarkan frekuensi kemunculannya kemudian diurutkan secara *ascending*. Tabel 3.10 menunjukkan frekuensi kemunculan setiap simbol.

Table 3.10 Tabel Frekuensi

Simbol	Frekuensi
h	2
a	2
d	1
i	1
\n	1
\r	1

Setiap simbol yang memiliki frekuensi rendah di proses terlebih dahulu. Proses pembentukan pohon Huffman ditunjukkan pada gambar 3.20.



Gambar 3.20 Pembentukan Pohon Huffman

Langkah selanjutnya adalah membentuk tabel kode Huffman berdasarkan pohon Huffman. Tabel kode Huffman ditunjukkan pada tabel 3.11.

Table 3.11 Enkoding Kode Huffman

Simbol	Kode Huffman
h	10
a	11
d	010
i	011
\n	000
\r	001

Hasil dari enkoding ‘hadiah’ berdasarkan tabel 3.11 sebagai berikut :

hadiah = 10110100111110000001

B. Enkripsi DES

Hasil enkoding dari kompresi Huffman berikutnya di enkripsi dengan DES. Apabila panjang hasil enkoding kurang dari kelipatan 64 maka harus diberi bit tambahan berupa ‘0’.

Setelah mengalami penambahan bit ‘0’ :
1011010011111000000100000000000000000000000000000000000000000000

Proses enkripsi DES dimulai dari pembentukan kunci internal. Kunci yang ekstenal ‘sync458r’ dirubah dahulu ke bentuk biner. Tabel biner kunci ditunjukkan pada tabel 3.12. Berdasarkan kunci eksternal yang telah dijadikan biner tersebut, selanjutnya akan dilewatkan melalui tabel permutasi *PC-1* yang telah dijelaskan pada tabel 2.1. Pada permutasi *PC-1*, setiap bit kedelapan dari kunci akan diabaikkan, sehingga akan menghasilkan panjang kunci eksternal yang semula 64-bit sekarang menjadi 56-bit. Hasil dari permutasi *PC-1* tersebut ditunjukkan pada tabel 3.13.

Table 3.12 Biner Kunci

sync458r	0	1	1	1	0	0	1	1
	0	1	1	1	1	0	0	1
	0	1	1	0	1	1	1	0
	0	1	1	0	0	0	1	1
	0	0	1	1	0	1	0	0
	0	0	1	1	0	1	0	1
	0	0	1	1	1	0	0	0
	0	1	1	1	0	0	1	0

Table 3.13 Hasil Permutasi *PC-1*

0	0	0	0	0	0	0
0	1	0	0	0	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	0	0	0	1	1	0
1	0	0	1	1	0	1
0	0	0	1	0	0	0
1	1	0	0	0	1	1

Kunci eksternal yang sudah mengalami permutasi *PC-1* berikutnya dibagi menjadi dua kelompok yakni C_0 dan D_0 yang masing-masing memiliki panjang 28-bit.

C_0 = 0000000 0100011 1111111 1111111
D_0 = 1000110 1001100 0001000 0100011

C_0 dan D_0 mengalami pergeseran ke kiri (*left shift*) sesuai jumlah bit yang telah ditentukan pada tabel 2.2 Pergesaran ini dilakukan sebanyak 16 kali putaran, sehingga akan diperoleh nilai $C_{16} = C_0$ dan $D_{16} = D_0$. Hasil dari tiap putarannya ditunjukkan sebagai berikut :

Putaran 1

C_1	D_1
00000001000111111111111111110	00011010011010001000110001111

Putaran 2

C_2	D_2
00000010001111111111111111100	00110100110100010001100011110

Putaran 3

C_3	D_3
00001000111111111111111110000	11010011010001000110001111000

Putaran 4

C_4	D_4
00100011111111111111111000000	01001101000100011000111000111

Putaran 5

C_5	D_5
10001111111111111111100000000	00110100010001100011100011101

Putaran 6

C_6	D_6
00111111111111111110000000010	11010001000110001110000110100

Putaran 7

C_7	D_7
11111111111111111000000001000	01000100011000111000111010011

Putaran 8

C_8	D_8
11111111111111100000000100011	00010001100011100011101001101

Putaran 9

C_9	D_9
11111111111110000000001000111	00100011000111000110100110 10

Putaran 10

C_{10}	D_{10}
11111111111000000001000111 11	10001100011100011010011010 00

Putaran 11

C_{11}	D_{11}
11111111100000000100011111111	00110001110001101001101000 10

Putaran 12

C_{12}	D_{12}
11111110000000010001111111111	11000111000110100110100010 00

Putaran 13

C_{13}	D_{13}
11111000000001000111111111111	00011100011010011010001000 11

Putaran 14

C_{14}	D_{14}
11100000000100011111111111111	01110001101001101000100011 00

Putaran 15

C_{15}	D_{15}
10000000010001111111111111111	11000110100110100010001100 01

Putaran 16

C_{16}	D_{16}
00000000100011111111111111111	10001101001101000100011000 11

Dari setiap putaran C_i dan D_i, berikutnya dilewatkan melalui tabel *PC-2* untuk mendapatkan kunci internal K_1, K_2, K_n , K_{16}. Tabel *PC-2* dapat dilihat pada tabel 2.5. Dari proses PC-2 tersebut diperoleh kunci internal K_i dengan panjang 48-bit. Hasil dari pembentukan kunci intenal K_i ditunjukkan sebagai berikut :

Key 1	1	1	0	1	0	0
	0	0	1	0	1	0
	1	1	1	0	1	1
	1	0	1	1	1	0
	1	0	0	0	0	1
	0	0	0	1	1	0
	0	1	1	1	0	0
	0	0	1	0	0	1

Key 2	1	1	1	1	0	0
	0	0	1	0	1	0
	1	1	1	0	1	0
	1	1	0	1	1	0
	0	0	1	1	0	1
	0	1	0	0	0	1
	1	1	0	0	1	1
	0	0	0	0	0	1

Key 3	1	1	1	1	0	1
	0	0	1	0	1	1
	1	1	1	0	0	0
	1	0	0	1	1	0
	0	1	0	0	1	0
	1	0	1	0	0	0
	0	0	0	0	0	1
	1	1	0	1	1	1

Key 4	1	1	1	0	0	0
	1	0	1	0	1	1
	0	1	1	0	0	1
	1	1	0	1	1	0
	0	0	0	0	0	1
	1	1	1	1	1	0
	1	1	0	1	1	0
	0	0	1	1	0	0

Key 5	1	1	1	0	1	1
	0	0	1	1	0	1
	0	1	1	0	0	1
	1	1	0	1	1	0
	0	0	1	0	1	0
	0	0	0	0	0	1
	0	1	0	1	1	1
	0	1	0	0	0	1

Key 6	1	1	1	0	0	1
	1	0	1	1	0	1
	0	0	1	1	0	1
	1	1	1	0	1	0
	1	1	0	0	1	0
	1	1	1	1	0	0
	0	0	0	0	0	0
	1	0	0	1	1	1

Key 7						
	1	0	1	0	1	1
	1	0	1	1	0	1
	0	0	1	1	0	1
	1	1	0	0	1	1
	0	1	0	0	0	1
	1	0	0	1	0	0
	1	1	1	1	1	0
	0	0	1	0	0	0

Key 8						
	1	0	1	0	1	1
	1	1	0	1	0	1
	1	0	1	1	0	1
	0	1	1	0	1	1
	1	0	0	1	1	0
	0	0	0	0	0	1
	0	0	0	1	0	1
	0	1	1	1	0	1

Key 9						
	0	0	1	0	1	1
	1	1	0	1	0	1
	0	0	1	1	1	1
	0	1	1	0	1	1
	1	1	1	0	1	1
	0	1	0	0	0	0
	0	1	0	0	0	1
	1	1	0	1	0	0

Key 10						
	0	0	1	1	1	1
	1	1	0	1	0	1
	0	0	0	1	1	1
	0	1	1	1	0	1
	0	0	0	0	1	0
	0	1	1	1	1	0
	1	0	1	0	1	1
	0	0	1	0	1	0

Key 11						
	0	0	0	1	1	1
	1	1	0	1	0	0
	1	0	0	1	1	1
	0	1	1	0	0	1
	0	0	1	1	0	1
	0	0	1	1	0	1
	0	1	0	0	0	0
	0	1	0	1	0	1

Key 12						
	0	0	0	1	1	1
	1	1	0	1	1	0
	1	0	0	1	1	0
	1	1	1	1	0	1
	1	0	0	0	1	0
	1	1	0	0	0	0
	0	1	0	0	1	1
	1	0	0	1	1	0

Key 13	1	0	0	1	1	1
	1	1	0	0	1	0
	1	1	0	1	1	0
	0	0	1	1	0	1
	1	0	0	0	1	1
	0	0	1	1	1	0
	1	0	1	1	1	0
	0	0	0	0	0	1

Key 14	0	1	0	1	1	0
	1	1	0	0	1	0
	1	1	1	0	1	0
	1	0	1	1	0	1
	0	0	1	1	0	0
	1	0	0	1	0	0
	0	1	1	0	0	1
	0	1	0	1	0	1

Key 15	1	1	0	1	1	0
	0	1	1	0	1	1
	1	1	0	0	1	0
	1	0	1	1	0	0
	1	1	0	1	1	0
	1	1	1	0	0	0
	0	0	0	1	1	0
	0	0	0	0	1	0

Key 16	1	1	0	1	0	0
	0	1	1	0	1	0
	1	1	0	0	1	0
	1	0	1	1	1	0
	0	0	0	0	0	1
	0	1	1	0	1	0
	0	0	1	1	1	0
	1	1	1	1	1	0

Untuk menyandiakan kode Huffman tersebut pertama harus dilewatkan melalui permutasi awal (*IP*) sesuai tabel 2.4. Hasil dari permutasi awal ditunjukkan pada tabel 3.14.

Table 3.14 Hasil Permustasi Awal

IP							
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0

Hasil permutasi awal dibagi menjadi dua bagian L_0 dan R_0 dengan panjang masing-masing 32-bit. Setiap putaran dalam menyandiakan plainteks merupakan jaringan Feistel yang ditunjukkan pada persamaan 2.2. Sebagai contohnya untuk membentuk L_1 dan R_1 maka pertama kali R_0 akan diekpansi berdasarkan tabel 2.7 sehingga diperolah hasil ekpansi dengan panjang 48-bit yang ditunjukkan pada tabel 3.15.

Table 3.15 Ekspansi terhadap R_0

$E(R_0)$					
0	0	0	0	0	0
0	0	0	1	1	0
1	0	0	0	0	0
0	0	0	1	1	0
1	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Sesuai dengan persamaan 2.3, hasil ekpansi tersebut kemudian di XOR-kan dengan kunci internal K_1.

$E(R_0)$							K_1							Vektor A					
0	0	0	0	0	0		1	1	0	1	0	0		1	1	0	1	0	0
0	0	0	1	1	0		0	0	1	0	1	0		0	0	1	1	0	0
1	0	0	0	0	0		1	1	1	0	1	1		0	1	1	0	1	1
0	0	0	1	1	0	⊕	1	0	1	1	1	0	=	1	0	1	0	0	0
1	0	0	0	0	0		1	0	0	0	0	1		0	0	0	0	0	1
0	0	0	1	0	0		0	0	0	1	1	0		0	0	0	0	1	0
0	0	0	0	0	0		0	1	1	1	0	0		0	1	1	1	0	0
0	0	0	0	0	0		0	0	1	0	0	1		0	0	1	0	0	1

Dari proses tersebut terbentuklah vektor *A* yang akan menjadi masukkan bagi setiap kotak-*S*. Vektor *A* dibagi menjadi 8

kelompok, dimana setiap kelompok memiliki panjang 6-bit. Dari setiap kelompok dicari indeks baris dan kolomnya. Misalkan saja pada S_1 yang sudah terbentuk dari vektor A tesebut sebagai berikut :

1	1	0	1	0	0

Keterangan

Indek baris

Indek kolom

Indeks baris menunjukkan nilai biner 10 kemudian dirubah kebentuk desimal menjadi 2. Pada indeks kolom memiliki nilai biner 1010 yang jika dirubah ke desimal menjadi 10. Dari indeks baris dan kolom menunjukkan koordinat dari nilai S_1 yang ditunjukkan pada tabel 3.12. Dari hasil perpotongan tersebut diperoleh nilai desimal 9, yang selanjutnya dirubah ke biner menjadi 1001. Hal ini dilakukan sampai kotak-*S* yang ke-8 yang ditunjukkan pada tabel 3.16 sampai 3.23.

Table 3.16 Kotak-S_1 (110100)

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_1 (110100) = 1001

Table 3.17 Kotak-S_2 (001100)

S_2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_2 (001100) = 0011

Table 3.18 Kotak-S_3 (011011)

S_3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_3 (011011) = 1011

Table 3.19 Kotak-S_4 (101000)

S_4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_4 (101000) = 1100

Table 3.20 Kotak-S_5 (000001)

S_5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	7	14	2	13	6	15	0	9	10	4	5	3

S_5 (000001) = 1110

Table 3.21 Kotak-S_6 (000010)

S_6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_6 (000010) = 0001

Table 3.22 Kotak-S_7 (011100)

S_7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	3	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	7	5	0	15	14	2	3	12

S_7 (011100) = 0110

Table 3.23 Kotak-S_8 (001001)

S_8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S_8 (001001) = 1010

Hasil dari S_1 sampai S_8 dipermutasi dengan matrik permutasi P sesuai tabel 2.16 untuk menghasilkan vektor B. Langkah akhir untuk mendapatkan R_1 adalah L_0 di XOR-kan dengan vektor B.

L_0					$P_{(B)}$					R_1			
0	0	0	0		0	1	0	0		0	1	0	0
0	0	1	0		1	1	0	1		1	1	1	1
0	0	0	0		1	0	0	1		1	0	0	1
0	1	1	1	$\oplus$	0	1	1	0	=	0	0	0	1
0	0	0	0		0	1	1	1		0	1	1	1
0	0	0	1		0	1	0	1		0	1	0	0
0	0	0	0		1	1	0	0		1	1	0	0
0	0	0	0		0	1	1	0		0	1	1	0

Berdasarkan persamaan 2.2 maka nilai L_1 diperoleh dari R_{1-1}. Sehingga dalam putaran ke-*1* diperoleh nilai :

L_1R_1	00000011000000110000001000000000 01001111100100010111010011000110

Untuk selanjutnya dilakukan perputaran sebanyak 16 kali sesuai jaringan Feistel, sehingga diperoleh nilai akhir L_{16}, R_{16}.

L_2R_2	01001111100100010111010011000110 00000010000100000110100001100110
L_3R_3	00000010000100000110100001100110 10001101110111011101111101110010
L_4R_4	10001101110111011101111101110010 00110000000000011010100011011010
L_5R_5	00110000000000011010100011011010 00100101110111100100110011000011
L_6R_6	00100101110111100100110011000011 11110010011111101110101111011000
L_7R_7	11110010011111101110101111011000 01010110101101000000001000011000
L_8R_8	01010110101101000000001000011000 11011111111001110111110101000011
L_9R_9	11011111111001110111110101000011 00101010011011011100010110111111
$L_{10}R_{10}$	00101010011011011100010110111111 01100011101111011001101111011111

$L_{11}R_{11}$	01100011101111011001101111011111 01110111101110111111010100100101
$L_{12}R_{12}$	01110111101110111111010100100101 00101000110010110010111101100010
$L_{13}R_{13}$	00101000110010110010111101100010 01011001111100101111100101000010
$L_{14}R_{14}$	01011001111100101111100101000010 01100110110010110110110001001001
$L_{15}R_{15}$	01100110110010110110110001001001 01100100001110001001110001010001
$L_{16}R_{16}$	01100100001110001001110001010001 10011110111011101010100100110111

Tahap akhir dalam pembentukan cipherteks adalah melakukan invers permutasi (IP^{-1}) pada L_{16}, R_{16} sesuai pada tabel 2.17.

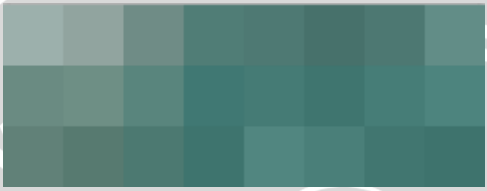
$R_{16}L_{16}$							
1	0	0	1	1	1	1	0
1	1	1	0	1	1	1	0
1	0	1	0	1	0	0	1
0	0	1	1	0	1	1	1
0	1	1	0	0	1	0	0
0	0	1	1	1	0	0	0
1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	1

IP^{-1}

Cipherteks							
0	0	0	0	0	1	1	1
0	1	0	1	0	0	0	1
1	1	0	1	1	0	0	1
0	1	1	1	1	1	0	0
0	1	1	0	1	0	1	1
1	0	1	1	0	1	0	1
1	0	0	1	0	0	1	0
0	1	0	1	1	1	0	0

C. Penyisipan LSB

Penyisipan LSB dilakukan dengan cara mengganti tiap bit terakhir dari RGB per piksel dengan biner dari cipherteks. Misalkan saja berkas citra penampung berformat bitmap 24-bit dengan ukuran 8x3 piksel yang ditunjukkan pada gambar 3.23.



Gambar 3.21 Bitmap 24-bit 8x3 piksel

Berdasarkan gambar 3.21 di dapatkan representasi biner RGB tiap pikselnya yang ditunjukkan pada tabel 3.24.

Table 3.24 Representasi Biner RGB Citra Penampung

Piksel	R	G	B
1	10011100	10110000	10101100
2	10010001	10100100	10011111
3	01101111	10001100	10000110
4	01010000	01111101	01110110
5	01001110	01111001	00000000
6	01000111	01110001	01101011
7	01001101	01111000	01110010
8	01100010	10001101	10000111
9	01101010	10001011	10000010
10	01101110	10001111	10000101
11	01011001	10000101	01111101
12	01000000	01111000	01110011
13	01000101	01111011	01110101
14	00111111	01110101	01101111
15	01000110	01111101	01110111

16	01001101	10000100	01111110
17	01100001	10000001	01111000
18	01010111	01111010	01110000
19	01001100	01111001	01110001
20	00111110	01110100	01101110
21	01010001	10000110	10000000
22	01001010	01111111	01111001
23	01000001	01110110	01110000
24	00111110	01110011	01101101

Cipherteks disisipkan ke dalam citra penampung dengan menggantikan bit terakhir tiap RGB. Dari hasil penyisipan inilah citra stego yang berisi pesan rahasia dihasilkan. Tabel 3.25 menunjukkan representasi dari citra stego.

Table 3.25 Representasi Biner RGB Citra Stego

Piksel	R	G	B
1	1001110**0**	1011000**0**	1010110**0**
2	1001000**0**	1010010**0**	1001111**1**
3	0110111**1**	1000110**1**	1000011**0**
4	0101000**1**	0111110**0**	0111011**1**
5	0100111**0**	0111100**0**	0000000**0**
6	0100011**1**	0111000**1**	0110101**1**
7	0100110**0**	0111100**1**	0111001**1**
8	0110001**0**	1000110**0**	1000011**1**
9	0110101**0**	1000101**1**	1000001**1**
10	0110111**1**	1000111**1**	1000010**1**
11	0101100**0**	1000010**0**	0111110**0**
12	0100000**1**	0111100**1**	0111001**0**
13	0100010**1**	0111101**0**	0111010**1**
14	0011111**1**	0111010**1**	0110111**0**
15	0100011**1**	0111110**1**	0111011**0**
16	0100110**1**	1000010**0**	0111111**1**
17	0110000**1**	1000000**0**	0111100**0**

18	01010111	01111010	01110000
19	01001101	01111000	01110000
20	00111111	01110100	01101111
21	01010001	10000111	10000000
22	01001010	01111111	01111001
23	01000001	01110110	01110000
24	00111110	01110011	01101101

3.5.2 Perhitungan Manual Pada Penguraian

A. Ekstraksi LSB

Proses perhitungan manual pada ekstraksi LSB merupakan kebalikan dari proses penyisipan. Langkah pertama pada aplikasi ini adalah membaca SecrecKey.ser untuk mencari panjang dari cipherteks yang disisipkan. Berikutnya untuk menguraikan citra stego tersebut diambil bit-bit terakhir tiap piksel sampai panjang dari bit cipherteks. Misalkan saja pada tabel 3.24 diambil bit terakhir sampai 64-bit maka akan diperoleh sebagai berikut :

0000011101010001110110010111110001101011101101011001001001011100

B. Dekripsi DES

Hasil dari ekstraksi LSB yang berupa cipherteks selanjutnya di dekripsi dengan cara membalik kunci intenal DES. Kunci internal dimulai dari K_{16}, K_{15},... sampai K_1. Dalam setiap putaranya juga menggunakan fungsi dari jaringan Feistel, maka hasilnya sebagai berikut :

1011010011111000000100000000000000000000000000000000000000000000

C. Dekompresi Huffman

Rangkaian bit terkompresi kode Huffman yang diperoleh dari dekripsi di dekompresi untuk memperoleh kembali plainteks. SecretKey.ser yang menyimpan panjang dari bit kompresi di *load* terlebih dahulu. Hal ini dilakukan untuk menghilangkan bit yang tidak digunakan pada waktu kompresi. Berikutnya table.txt yang yang memuat tabel kode Huffman di *load*. Dekoding dilakukan mulai dari depan sampai belakang. Tabel 3.26 menunjukkan hasil dari dekoding pada dekompresi Huffman.

Table 3.26 Dekoding Kode Huffman

Biner terkompresi	Kode Huffman	Simbol
10	10	h
11	11	a
010	010	d
011	011	i
11	11	a
10	10	h
000	000	\n
001	001	\r

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang dijelaskan dalam sub bab ini meliputi lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pembuatan aplikasi *Advanced Steganography* ini menggunakan Notebook dengan spesifikasi sebagai berikut :

1. Processor Intel(R) Core(TM)2 Duo CPU T6400 @ 2.00GHz (2 CPUs), ~2.0GHz.
2. Memory 2048MB RAM.
3. Harddisk 320GB.
4. VGA Mobile Intel(R) 4 Series Express Chipset Family.

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan untuk mendukung pembuatan aplikasi Advance Steganografi ini meliputi :

1. Sistem Operasi Windows 7 32-bit.
2. Netbeans 7.1.
3. Notepad++.
4. Adobe Photoshop CS5.

4.2 Implementasi Program

Berdasarkan analisis dan perancangan yang telah dijelaskan pada bab 3, maka pada sub bab ini akan dijelaskan proses pengimplementasiannya. Implementasi program yang terbentuk terdiri dari 7 package java yang ditunjukkan pada tabel 4.1.

Table 4.1 Package

Fungsi	Kegunaan
des	Package yang berisi berkas *.java untuk proses enkripsi dan dekripsi DES
gui	Package yang berisi berkas *.java untuk pemrosesan GUI
huffman	Package yang berisi berkas *.java untuk proses kompresi dan dekompresi Huffman
lsb	Package yang berisi berkas *.java untuk proses penyisipan dan penguraian steganografi LSB
main	Package yang berisi berkas *.java untuk pemrosesan utama yaitu pada pengecekan plainteks, citra penampung, dan kunci.
utils	Package yang berisis berkas *.java untuk *utility* tambahan dalam hal konversi dan perhitungan PSNR.
widget	Package yang berisi berkas *.java untuk keperluan komponen *swing* dari GUI yang dipakai.

4.2.1 Implementasi Memuat Pesan

Untuk memuat pesan, dikerjakan oleh kelas filePlain.java yang ada di package *main*. Terdapat dua fungsi yang digunakannya yang ditunjukkan pada tabel 4.2.

Table 4.2 Fungsi Memuat Pesan

Fungsi	Kegunaan
`public filePlain(String sourcePlain)`	Konstruktor untuk mengolah string pesan.
`public String getPlain()`	Mengembalikan pesan yang sudah dibaca.

a. filePlain

FilePlain merupakan konstruktor untuk proses pembacaan plainteks. Terdapat satu parameter berupa *string* dari nama berkas yang akan dilakukan pembacaan. Proses baca berkas ditunjukkan pada *Source code* 4.1.

```
public filePlain(String sourcePlain) {
File file = new File(sourcePlain);
StringBuilder contents = new StringBuilder();
BufferedReader reader = null;
try {
reader = new BufferedReader(new FileReader(file));
String text;
while ((text = reader.readLine()) != null) {
contents.append(text).append(System.getProperty(
"line.separator"));
            }
        } catch (FileNotFoundException e) {
            System.err.println(e);
        } catch (IOException e) {
            System.err.println(e);
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
                System.err.println(e);
            }
        }
        plain=contents.toString();
        if(plain.length()%8!=0){
            lengthNow=8-plain.length()%8;
            while(lengthNow!=0){
                plain=plain+" ";
                lengthNow--;
            }
        }
    }
```

Source code 4.1 FilePlain

b. getPlain

GetPlain merupakan fungsi untuk mengembalikan nilai dari pesan yang sudah dibaca. Proses getPlain ditunjukkan pada *Source code* 4.2.

```
public String getPlain() {
    return plain;
}
```

Source code 4.2 GetPlain

4.2.2 Implementasi Kompresi Huffman

Sesuai dengan alur dari aplikasi ini, langkah awal yang perlu dikerjakan adalah melakukan proses kompresi Huffman terhadap plainteks. Terdapat empat fungsi yang paling penting pada kompresi Huffman yang ditunjukkan tabel 4.3.

Table 4.3 Fungsi Kompresi Huffman

Fungsi	Kegunaan
`sort(dataList)`	Mengurutkan dataList secara *ascending*
`makeTable()`	Membuat tabel Huffman
`makeTree()`	Membuat pohon Huffman
`saveTable()`	Menyimpan table Huffman
`encoding()`	Enkoding terhadap pesan

Pada proses sebelumnya, pesan telah dibaca dan dibalikkan kembali nilainya oleh berkas filePlain.java. Dari pembalikan nilai inilah plainteks tadi dimasukkan ke dalam *linkedlist*. *Linkedlist* merupakan tipe data yang mampu menyimpan data dan *pointer*. Data yang disimpan pada *linkedlist* ini merupakan simbol, frekuensi, serta kode Huffman-nya. Proses ini dikerjakan oleh berkas compress.java yang ada di package huffman yang ditunjukkan pada *Source code* 4.3.

```
public String compression(String plainteks) {
    token=plainteks.toCharArray();
    for(int i=0;i<plainteks.length();i++){
        addToList(token[i]);
    }
    sort(dataList);
    makeTable();
    makeTree();
    saveTable();
    encoding();
    return compressCode;
}
```

Source code 4.3 Compress

a. sort

Sort merupakan proses pengurutan secara *ascending* terhadap datalist. Datalist dengan frekuensi kemunculan terendah akan berada di atas, begitu juga sebaliknya. Proses sort ditunjukkan pada *Source code* 4.4.

```
private void sort(dataList dataList) {
    Collections.sort(dataList,dataCompare);
}
```

Source code 4.4 Sort

b. makeTable

MakeTable merupakan proses yang digunakan untuk membuat table Huffman yang ditunjukkan pada *Source code* 4.5. Proses pembuatan tabel dengan cara memberikan inisialisasi *true* atau *false* pada setiap percabangan. *True* berarti pohon sudah dijadikan satu akar sedangkan *false* adalah kebalikannya. Proses tersebut akan diulang terus sampai menghasilkan satu akar.

```
private void makeTable() {
    dataList inList=(dataList) dataList.clone();
    dataList temp;
    int size=inList.size();
    outList.add(inList.clone());
    while(size>2){
        ((data)inList.get(size-2)).append(((data)
        inList.get(size-1)));
        inList.removeLast();
```

```
            temp=(dataList) inList.clone();
            ((data)inList.get(size-2)).setFlag(true);
            setToflagOthers(temp);
            sort(inList);
            sort(temp);
            outList.add(temp);
            size--;
        }
}
```

Source code 4.5 MakeTable

c. makeTree

MakeTree digunakan untuk membuat pohon Huffman. Proses ini sangat sederhana yaitu dengan menelusuri setiap simbol dan membandingkannya. Untuk *node* sebelah kiri diberi label '0' dan *node* kanan label '1'. Proses makeTree ditunjukkan pada *Source code* 4.6.

```
private void makeTree() {
    dataList temp1=(dataList)
outList.get(outList.size()-1);
    dataList temp2;
    ((data)temp1.get(0)).setBinaryCode("0");
    ((data)temp1.get(1)).setBinaryCode("1");
    for(int i=outList.size()-1;i>0;i--){
        temp1=(dataList) outList.get(i);
        temp2=(dataList) outList.get(i-1);
        compareCode(temp1,temp2);
    }
}
```

Source code 4.6 MakeTree

d. saveTable

Tabel Huffman yang telah dibentuk selanjutnya disimpan dalam table.txt. Penyimpanan table ini digunakan pada waktu enkoding ataupun dekoding. Tabel Huffman yang disimpan pada aplikasi ini bukan karakter dan kode Huffmannya, melainkan kode ASCII dari karakter dan kode Huffmannya. Proses dari penyimpanan tabel ini ditunjukkan pada *Source code* 4.7.

```
private void saveTable(){
try{
bufWrite=new BufferedWriter
         (new FileWriter("table.txt"));
         dataList table=(dataList) outList.get(0);
         for(int i=0;i<table.size();i++){
         bufWrite.write(((data)table.get(i))
         .getCharCode()+";"+((data)table
         .get(i)).getBinaryCode()+";");
             bufWrite.newLine();
         }
         bufWrite.flush();
         bufWrite.close();
    }catch(IOException e){
       System.out.println("Penulisan table error");
    }
}
```

Source code 4.7 SaveTable

e. encoding

Encoding merupakan proses menyandikan plainteks dengan kode Huffman yang telah terbentuk. Proses ini mengacu pada table Huffman yang sebelumnya sudah disimpan. Proses enkoding ditunjukkan pada *Source code* 4.8.

```
private void encoding() {
    int codeHuffman;
    for(int i=0;i<token.length;i++){
        codeHuffman=(int)(token[i]);
        compressCode+=getCodeBinary(
        String.valueOf(codeHuffman));
    }
 }
```

Source code 4.8 Encoding

4.2.3 Implementasi Enkripsi DES

DES akan mengenkripsi plainteks dimana plainteksnya berupa kode Huffman yang sebelumnya telah dibentuk. Ciri dari kriptografi DES adalah menyandikan plainteks setiap kelipatan 64-bit. Apabila panjang dari kode Huffman kurang dari kelipatan 64-bit dilakukan penambahan bit ‘0’. Terdapat lima fungsi yang paling penting pada enkripsi DES yang ditunjukkan pada tabel 4.4. Hasil

dari proses enkripsi ini akan mengembalikan nilai *string* dari cipherteks untuk diproses lebih lanjut pada steganografi.

Table 4.4 Fungsi Enkripsi DES

Fungsi	Kegunaan
`checkPlain(plain)`	Memastikan panjang dari plainteks
`generate(key)`	Membuat kunci internal untuk proses enkripsi dan dekripsi
`doIP(subPlain,outIP)`	Permutasi awal saat enkripsi pada subPlain
`generateEncrypt(L,R,keyOut)`	Proses perhitungan jaringan Feistel
`setInversIP(RL)`	Invers permutasi terhadap RL
`saveSec()`	Menyimpan panjang dari cipherteks

Secara keseluruhan proses utama enkripsi DES terdapat pada berkas encryption.java yang ada di package des. *Source code* 4.9 menunjukkan proses dari enkripsi DES.

```
public String encrypt(String key, String plain){
    this.plain=checkPlain(plain);
    feistel=new feistel();
    genKey=new generateKey();
    conCipher=new stringCipherToString();
    plainToBinary=new stringToBinary();
    keyOut=genKey.generate(key);
    int mulai=0;
    int akhir=64;
    for(int i=0;i<this.plain.length()/64;i++){
        subPlain=this.plain.substring(mulai,akhir);
        doIP(subPlain,outIP);
        L[0]=outIP.substring(0,outIP.length()/2);

    R[0]=outIP.substring(outIP.length()/2,
        outIP.length());
        RL=feistel.generateEncrypt(L,R,keyOut);
```

```
            setInversIP(RL);
            mulai=akhir;
            akhir+=64;
        }
        saveSec();
        return binaryCipher;
}
```

Source code 4.9 Encryption

a. checkPlain

CheckPlain digunakan untuk mengecek apakah panjang dari plainteks sudah kelipatan 64-bit atau belum. Jika plainteks masih belum kelipatan 64-bit, maka plainteks harus ditambahi bit '0'. Dari proses ini akan dikembalikan lagi nilai plainteks. *Source code* 4.10 menunjukkan proses dari checkPlain.

```
private String checkPlain(String plain) {
    if(plain.length()%64!=0){
        int lengthNow=64-plain.length()%64;
        while(lengthNow!=0){
            plain=plain+"0";
            lengthNow--;
        }
    }
    return plain;
}
```

Source code 4.10 CheckPlain

b. doIP

DoIP digunakan untuk melakukan permutasi awal terhadap blok plainteks. Terdapat dua parameter bertipe *string* yaitu binaryPlain dan outIP. BinaryPlain merupakan blok 64-bit plainteks, sedangkan outIP merupakan hasil dari proses permutasi awal tersebut. *Source code* 4.11 menunjukkan proses dari doIP.

```
public void doIP(String binaryPlain, String outIP) {
    outIP="";
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
 outIP+=binaryPlain.substring(IP[i][j]-1,IP[i][j]);
        }
    }
    this.outIP=outIP;
}
```

Source code 4.11 DoIP

c. generateEncrypt

GenerateEncrypt merupakan proses jaringan Feistel. Terdapat tiga parameter bertipe *array string* yakni L, R, dan keyOut. L adalah bagian kiri dari matrik hasil pembentukan permutasi awal, sedangkan R adalah bagian kanannya. Panjang dari masing-masing L dan R adalah 32-bit. Untuk KeyOut adalah *array* yang berisi kunci internal yang sudah di *generate*. Hasil dari generateEncrypt akan mengembalikan nilai gabungan dari R dan L. Proses generateEncrypt ditunjukkan pada *Source code* 4.12.

```
public String generateEncrypt(String[] L, String[] R,
String[] keyOut) {
    String RL="";
    for(int i=1;i<=16;i++){
        getExpanding(R[i-1]);
        getVektorA(tempR,keyOut[i-1]);
        getSbox(vektorA);
        getPB(vektorB);
        L[i]=R[i-1];
        R[i]=getR(L[i-1],outPB);
    }
    return RL=R[16].concat(L[16]);
}
```

Source code 4.12 GenerateEncrypt

d. setInversIP

Tahapan akhir dalam proses enkripsi adalah melakukan invers permutasi terhadap RL untuk menghasilkan cipherteks. Sourrce code 4.13 menunjukkan proses dari setInversIP.

```
private void setInversIP(String RL) {
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            binaryCipher+=RL.substring(
            inIP[i][j]-1,inIP[i][j]);
        }
    }
}
```

Source code 4.13 SetInversIP

4.2.4 Implementasi Penyisipan LSB

Proses penyisipan LSB terdiri dari 3 fungsi penting yang ditunjukkan pada tabel 4.5. Pertama dilakukan pengecekan terhadap citra penampung untuk mengetahui seberapa banyak bit pesan yang mampu disisipkan. Dari hasil pengecekan, langkah berikutnya adalah membaca cipherteks dan menyisipkannya ke dalam citra penampung.

Table 4.5 Penyisipan LSB

Fungsi	Kegunaan
`chCompress(BufferedImage bufImage, int length)`	Menghitung jumlah bit pesan yang mampu disisipkan pada citra penampung
`readcomCodeteks(String compressCode)`	Membaca cipherteks
`readcomCodeteks(String compressCode)`	Menyisipkan cipherteks ke dalam citra penampung

Proses penyisipan steganografi LSB ini dikerjakan oleh steganoInsert.java yang ditunjukkan pada *Source code* 4.14.

```
public void insert(String cipherChar,
    String sourceImage){
    comCode=new String[cipherChar.length()];
    try {
     bufImage=ImageIO.read(new File(sourceImage));
    } catch (IOException ex){
Logger.getLogger(steganoInsert.class.getName())
.log(Level.SEVERE, null, ex);
```

```
    }
    checkImagePlain chImage=new checkImagePlain();
    check=chImage.chCompress(bufImage,
    cipherChar.length());
    if(check==true){
        readcomCodeteks(cipherChar);
        replaceImage(bufImage,comCode);
    }
    else{
        System.exit(1);
    }
}
```

Source code 4.14 SteganoInsert

a. chCompress

ChCompress digunakan untuk menghitung seberapa banyak bit pesan yang mampu ditampung oleh citra penampung. Fungsi chCompress ditunjukkan pada *Source code* 4.15.

```
public boolean chCompress(BufferedImage bufImage, int
length){
    lengthBitPesan=bufImage.getWidth()*bufImage
    .getHeight()*3;
    if(length>lengthBitPesan){
        return cek=false;
    }else{
        return cek=true;
    }
}
```

Source code 4.15 ChCompress

b. readComCodeteks

ReadComCodeteks digunakan untuk memasukkan cipherChar yang berupa hasil cipherteks ke dalam *array* satu dimensi. Proses readComCodecteks ditunjukkan pada *Source code* 4.16.

```
private void readcomCodeteks(String cipherChar) {
    String temp;
    for(int i=0;i<cipherChar.length();i++){
        comCode[i]=cipherChar.substring(i,i+1);
    }
}
```

Source code 4.16 ReadComCodeteks

c. replaceImage

ReplaceImage merupakan proses untuk menggantikan bit terakhir tiap RGB dari setiap piksel dengan bit dari cipherteks. Proses replaceImage ditunjukkan oleh *Source code* 4.17.

```
public void replaceImage(BufferedImage bufImage,
String[] comCode) {
    integerToBinary=new integerToStringBinary();
    int choice=0;
    int x=0; int y=0;
    int red,green,blue;
    for(int i=0;i<comCode.length;i++){
        color=new Color(bufImage.getRGB(x,y));
        if(choice==0){
            stringRed=integerToBinary.getBinary(color
.getRed()).substring(0,7)+comCode[i];
            red=Integer.parseInt(stringRed,2);

bufImage.setRGB(x,y,red<<16|color.getGreen()<<8|
color.getBlue());
            choice++;
        }else if(choice==1){
            stringGreen=integerToBinary.getBinary(color
.getGreen()).substring(0,7)+comCode[i];
            green=Integer.parseInt(stringGreen,2);

bufImage.setRGB(x,y,color.getRed()<<16|green<<8|
color.getBlue());
            choice++;
        }else{
            stringBlue=integerToBinary.getBinary(color
.getBlue()).substring(0,7)+comCode[i];
            blue=Integer.parseInt(stringBlue,2);
bufImage.setRGB(x,y,color.getRed()<<16|
color.getGreen()<<8|blue);
            x++;
            if(x==bufImage.getWidth()){
```

```
                    y++;
                    x=0;
                }
                choice=0;
            }
        }
        this.bufImage=bufImage;
        drawBuffer(bufImage);
}
```

Source code 4.17 ReplaceImage

4.2.5 Implementasi Memuat Berkas Rahasia

Untuk melakukan proses penguraian pada aplikasi ini yaitu dengan membaca isi dari berkas Secret.ser. Dalam berkas Secret.ser terdapat informasi mengenai panjang dari kode Huffman dan panjang dari cipherteks yang ada pada citra stego. Proses pembacaan pada Secret.ser ditunjukkan pada *Source code* 4.18.

```
public fileSecret(){
try{
bufReader=new BufferedReader(
        new FileReader("SecretKey.ser"));
        while((text=bufReader.readLine())!=null){
            count=0;
            pos=-1;
            for(int i=0;i<text.length();i++){
                char content=text.charAt(i);
                if(content==qoute){
                    if(count==0){
                compressLength=Integer
                .parseInt(text.substring(pos+1,i));
                        pos=i;
                        count++;}
else if(count==1){ cipherLength=Integer
            .parseInt(text.substring(pos+1,i));
                        pos=i;
                        count++;}
                }
            }
        }
    }catch(IOException e){ System.out.println(e);
    }
}
```

Source code 4.18 FileSecret

4.2.6 Implementasi Penguraian LSB

Panjang dari cipherteks telah diperoleh dari proses pembacaan berkas Secret.ser. Dengan batasan panjang cipherteks inilah bit-bit dari citra stego diambil. Proses dari penguraian LSB ditunjukkan pada *Source code* 4.19.

```
public String load(String sourceImage, int lengthCodec)
{
try {
     bufImage=ImageIO.read(new File(sourceImage));
    } catch (IOException ex) {
       Logger.getLogger(steganoLoad.class.getName())
.log(Level.SEVERE, null, ex);
    }
    integerToBinary=new integerToStringBinary();
    x=0;y=0;
    int choice=0;
    for(int i=0;i<lengthCodec;i++){
        c=new Color(bufImage.getRGB(x,y));
        if(choice==0){
            r=integerToBinary.getBinary(c.getRed())
            .substring(7,8);
            comCode=comCode+r;
            choice++;
        }else if(choice==1){
         g=integerToBinary.getBinary(c.getGreen())
         .substring(7,8);
         comCode=comCode+g;
         choice++;
        }else{b=integerToBinary.getBinary(c.getBlue())
.substring(7,8);
            comCode=comCode+b;
            r="";
            g="";
            b="";
            x++;
            if(x==bufImage.getWidth()){
                y++;
                x=0;
            }choice=0;
        }
    }return comCode;
}
```

Source code 4.19 steganoLoad

4.2.7 Implementasi Dekripsi DES

Source code pada dekripsi DES mirip dengan *source* pada enkripsinya. Hal yang membedakannya adalah penggunakan kunci internalnya. Pada proses dekripsi, kunci internal dimulai dari kunci yang ke-16. *Source code* 4.20 menunjukkan proses dari dekripsi DES.

```
public String decrypt(String key, String
cipherteksCode){
    String strBinary=cipherteksCode;
    feistel=new feistel();
    genKey=new generateKey();
    conCipher=new stringCipherToString();
    keyOut=genKey.generate(key);
    int mulai=0;
    int akhir=64;
    for(int i=0;i<strBinary.length()/64;i++){
        subCipher=strBinary.substring(mulai,akhir);
        doIP(subCipher,outIP);
        L[0]=outIP.substring(0,outIP.length()/2);

R[0]=outIP.substring(outIP.length()/2,outIP.length());
        RL=feistel.generateDecrypt(L,R,keyOut);
        setInversIP(RL);
        mulai=akhir;
        akhir+=64;
    }
    return tempPlain;
}
```

Source code 4.20 Decryption

4.2.8 Implementasi Dekompresi Huffman

Proses dekompresi dilakukan dengan cara membaca tabel kompresi kemudian membandingkannya. Terdapat dua fungsi yang ditunjukkan pada tabel 4.6.

Table 4.6 Dekompresi Huffman

Fungsi	Kegunaan
`readTable(new File("table.txt"))`	Membaca table.txt untuk menentukan panjang dari cipherteks
`isEqual(String subCodec)`	Membandingkan cipherteks dengan table.txt

Dekompresi Huffman dikerjakan oleh berkas decompress.java yang ditunjukkan pada *Source code* 4.21.

```
public String decompression(String codec, int
compressLength) {
    codec=codec.substring(0,compressLength);
    readTable(new File("table.txt"));
    int start=0;
    int sam;

    for(int i=1;i<=codec.length();i++){
        index=isEqual(codec.substring(start,i));
        if(index!=-1){
        charCode=Integer
        .parseInt(((data)listTable
        .get(index)).getCharCode());
            cipherteks+=(char)charCode;
            start=i;
        }
    }
    return cipherteks;
}
```

Source code 4.21 Decompress

a. readTable

ReadTable digunakan untuk proses pembacaan table.txt yang berisi kode Huffman dari kode ASCII dari setiap simbol. Proses readTable ditunjukkan pada *Source code* 4.22.

```
private void readTable(File table) {
    String text;
    int length=0;
    int count,pos;
    char quote=';';
    String codeKar="";
```

```
    String binaryCode="";
    try{
    bufReader=new BufferedReader(
    new FileReader(table));
       while((text=bufReader.readLine())!=null){
            count=0;
            pos=-1;
            for(int i=0;i<text.length();i++){
                char content=text.charAt(i);
            if(content==quote){
            if(count==0){
            codeKar=text.substring(pos+1,i);
                        pos=i;
                        count++;
             }else if(count==1){
             binaryCode=text.substring(pos+1,i);
                        pos=i;
                        count++;
                    }
                }
            }
            listTable.add(new
data(codeKar,binaryCode));
        }
    }catch(FileNotFoundException e){
        System.out.println(e);
    }catch(IOException e){
        System.out.println(e);
    }
}
```

Source code 4.22 ReadTable

b. isEqual

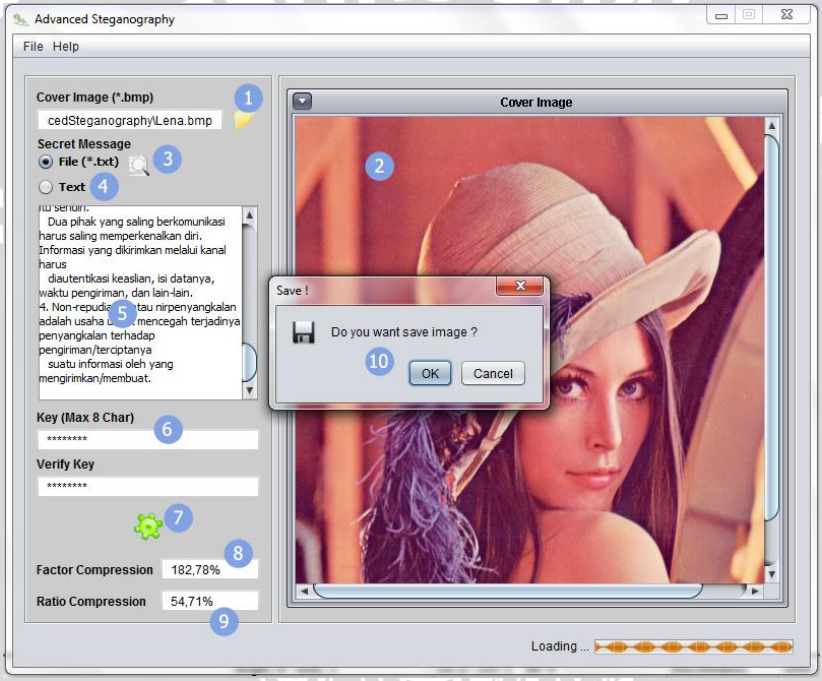
isEqual merupakan proses untuk membandingkan hasil dari cipherteks dengan tabel Huffman. Apabila hasilnya cocok, maka akan diambil simbol yang bersesuaian tersebut. Proses isEqual ditunjukkan pada *Source code* 4.23.

```
private int isEqual(String subCodec) {
    for(int i=0;i<listTable.size();i++)
if(((data)listTable.get(i)).getBinaryCode()
.equals(subCodec)){ return i; }
    }return -1;
}
```

Source code 4.23 IsEqual

4.3 Implementasi Antarmuka

Berdasarkan rancangan perangkat lunak yang telah dijelaskan pada bab 3, maka dibuatlah antarmuka seperti yang ditunjukkan pada gambar 4.1 , 4.2 dan 4.3. Pada gambar 4.1 ditunjukkan antarmuka untuk proses penyisipan. Gambar 4.2 menunjukkan antarmuka proses penguraian. Untuk gambar 4.3 merupakan antarmuka dari analisa MSE dan PSNR.



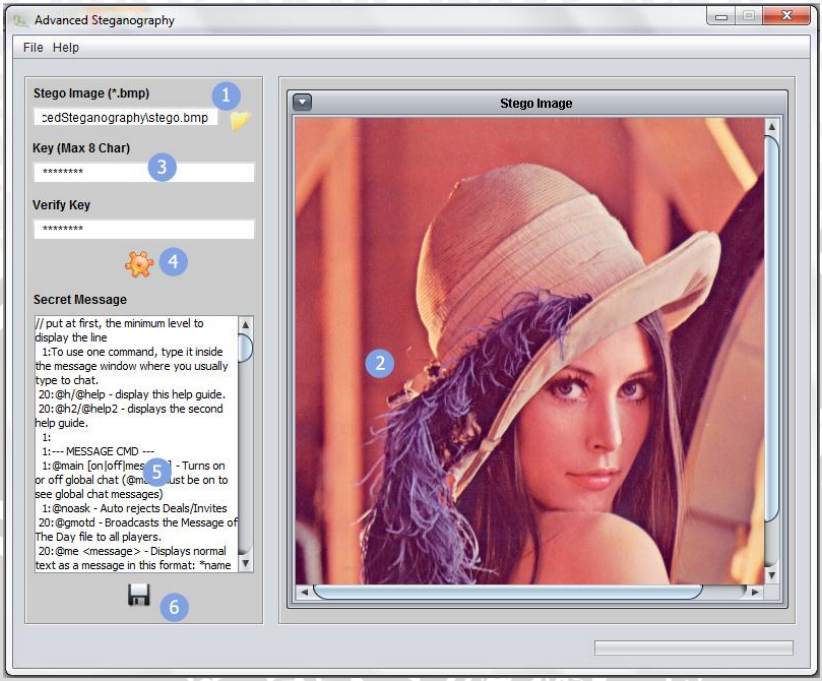
Gambar 4.1 Antaramuka Proses Penyisipan

Untuk melakukan proses penyisipan, *user* terlebih dahulu memasukkan citra penampung. Gambar dari citra penampung akan dimuat pada *internal frame* supaya mudah dilihat. Pesan yang disisipkan dapat berupa berkas *.txt atau *string* yang dimasukkan di *text area*. Proses pengamanan dilakukan oleh enkripsi DES, oleh karena itu *user* harus memasukkan kunci maksimal delapan karakter. Akan muncul *pop-up* peringatan apabila properti dari proses

penyisipan kurang. Nilai dari faktor kompresi dan rasio kompresi ditampilkan untuk mengetahui seberapa besar pesan mampu dikompresi. Setelah proses penyisipan berhasil dilakukan, akan muncul *pop-up* untuk menyimpan hasil dari citra stego. Berikut ini merupakan keterangan dari gambar 4.1 :

1. Tombol *open file* yang digunakan untuk membuka citra penampung berformat *.bmp 24-bit.
2. *Internal frame* yang digunakan untuk menampilkan citra penampung yang telah diseleksi.
3. Opsi untuk memilih pesan rahasia berupa berkas *.txt.
4. Opsi untuk memilih pesan rahasia berupa *string*.
5. *Text area* yang menampilkan isi dari pesan rahasia.
6. *Password field* yang digunakan untuk mengenkripsi pesan rahasia.
7. Tombol untuk proses penyisipan.
8. *Text field* yang berisi nilai dari faktor kompresi Huffman.
9. *Text field* yang berisi nilai dari rasio kompresi Huffman.
10. *Pop-Up* dialog untuk penyimpanan citra stego. *Pop-up* ini akan tampil ketika proses penyisipan pesan selesai dilakukan.

Untuk antarmuka proses penguraian, *user* disajikan tampilan yang kurang lebih sama seperti proses penyisipan. *User* diharuskan membuka berkas citra stego terlebih dahulu. Citra stego yang terseleksi akan ditampilkan pada *internal frame*. Berikutnya memasukkan kunci untuk proses dekripsi DES. Dilakukan pengecekan pada properti penguraian, jika properti belum lengkap maka akan muncul *pop-up* peringatan. Ketika semua properti penguraian lengkap maka proses penguraian dapat dilakukan dengan menekan tombol penguraian. Dari hasil penguraian tersebut, akan ditampilkan hasilnya pada *text area*. Tombol simpan yang berupa *icon* disket yang sebelumnya *disable* sekarang mejadi *enable*, sehingga *user* dapat menyimpan hasilnya dalam berkas berformat *.txt. Gambar 4.2 menunjukkan antarmuka dari proses penguraian.



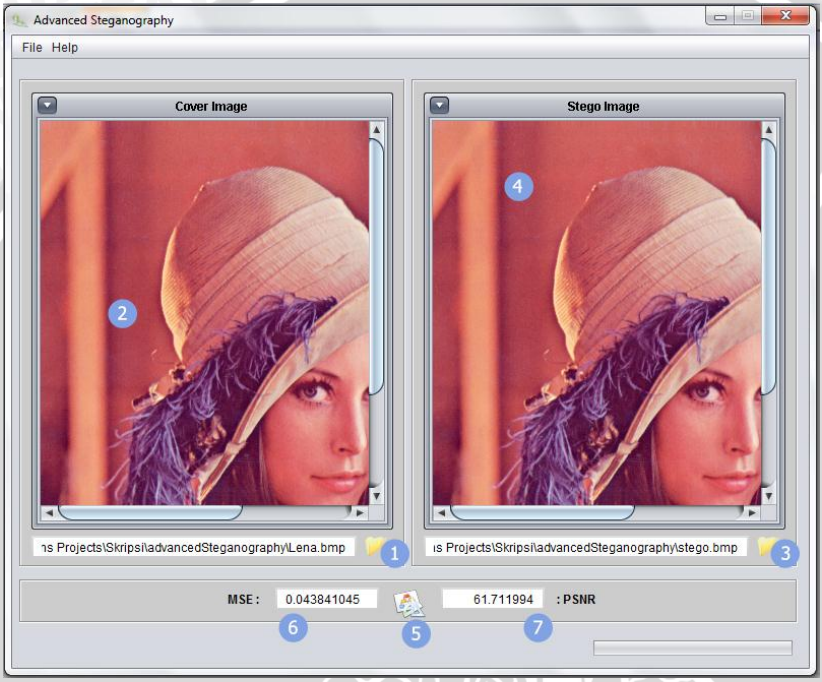
Gambar 4.2 Antarmuka Proses Penguraian

Keterangan pada gambar 4.2 sebagai berikut :

1. Tombol *open file* yang digunakan untuk membuka citra stego.
2. *Internal frame*, menampilkan citra stego yang telah diseleksi.
3. *Password field* yang berisi kunci untuk melakukan proses dekripsi.
4. Tombol proses penguraian.
5. *Text area* yang menampilkan hasil dari proses penguraian.
6. Tombol simpan yang digunakan untuk menyimpan hasil dari proses penguraian.

Dengan menganalisa derau antar citra penampung dengan citra stego maka dapat diketahui kualitas dari citra stego. Antarmuka proses analisa ditunjukkan pada gambar 4.3. *User* memasukkan citra penampung dengan menekan *open file* citra penampung, hasilnya akan ditampilkan dalam *internal frame cover image*. Citra stego

dimuat dengan menekan tombol *open file* citra stego, hasilnya dimuat dalam *internal frame stego image*. Berikutnya tinggal menekan tombol analisa untuk mengetahui seberapa besar nilai dari MSE dan PSNR.



Gambar 4.3 Antarmuka Proses Analisa

Keterangan dari gambar 4.3 sebagai berikut :

1. Tombol *open file* yang digunakan untuk membuka citra penampung.
2. *Internal frame* yang menampilkan citra penampung.
3. Tombol *open file* yang digunakan untuk membuka citra stego.
4. *Internal frame* yang menampilkan citra stego.
5. Tombol proses analisa.
6. *Text field* yang berisi nilai MSE dari proses analisa.
7. *Text field* yang berisi nilai dari PSNR dari hasil analisa.

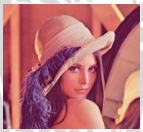
4.4 Implementasi Uji Coba

Pada sub bab berikut dilakukan pembahasan mengenai implementasi dari pengujian pada sistem serta ringkasan hasil pengujiannya.

4.4.1 Skenario Evaluasi

Terdapat tiga macam pengujian antara lain pengujian fungsionalitas perangkat lunak, kinerja perangkat lunak, dan ketahanan citra stego yang dihasilkan terhadap beberapa manipulasi citra. Citra penampung yang digunakan berformat *.bmp 24-bit. Detail dari citra yang akan diujikan ditunjukkan pada tabel 4.7.

Table 4.7 Daftar Citra Pengujian

No.	Nama Citra	Citra	Ukuran
1	Peppers.bmp		128x128 px
2	Baboon.bmp		256x256 px
3	Lena.bmp		512x512 px

Sesuai prosedur penelitian ini, pesan akan di kompres kemudian di enkripsi lalu dimasukkan ke dalam citra. Sebagai uji coba, pesan yang dimasukkan bervariasi ukurannya. Pada tabel 4.8 ditunjukkan tiga berkas *.txt sebagi pengujiannya.

Table 4.8 Daftar Berkas Pesan Rahasia

No.	Nama Berkas	Isi	Ukuran (bit)
1	Pesan1.txt	Kriptografi, secara umum adalah ilmu dan seni untuk menjaga (...)	13184
2	Pesan2.txt	// put at first, the minimum level to display the line 1:To use one command, type it (...)	22656
3	Pesan3.txt	- Additions in Wx interface (EternalHarvest) - Captcha support (Technology) - "party request PlayerName" (...)	71872

4.4.2 Hasil Pengujian

Hasil uji coba terhadap sistem dilakukan untuk mengetahui kinerja dari sistem yang telah dibangun. Pengujian yang dilakukan dibagi menjadi tiga sub bagian yakni pengujian fungsionalitas perangkat lunak, pengujian kinerja perangkat lunak, dan pengujian ketahanan citra stego yang dihasilkan.

4.4.2.1 Hasil Pengujian Fungsionalitas Perangkat Lunak

Pengujian fungsionalitas digunakan untuk mengetahui apakah semua fungsi yang digunakan dalam sistem dapat berjalan dengan benar. Pengujian ini meliputi pengujian fungsionalitas pada kode Huffman, DES, dan Steganografi.

Pada pengujian fungsionalitas kode Huffman, berkas dinyatakan berhasil apabila mampu di kompres dan di dekompres kembali ke bentuk plainteks. DES dinyatakan berhasil, apabila berkas dapat dienkripsi kemudian didekripsi kembali sehingga menghasilkan plainteks semula. Sedangkan pada Steganografi dinyatakan berhasil apabila berkas yang disisipkan tidak merusak citra penampung, serta dapat diuraikan kembali untuk menghasilkan

plainteks. Tabel 4.9, 4.10 dan 4.11 merupakan ringkasan dari hasil pengujian fungsionalitas.

Table 4.9 Pengujian Fungsionalitas Kode Huffman

Proses Kode Huffman			
No.	Nama Berkas	Kompresi	Dekompresi
1	Pesan1.txt	Berhasil	Berhasil
2	Pesan2.txt	Berhasil	Berhasil
3	Pesan3.txt	Berhasil	Berhasil

Table 4.10 Pengujian Fungsionalitas Kriptografi DES

Proses Kriptografi				
No.	Nama Berkas	Kunci	Enkripsi	Dekripsi
1	Pesan1.txt	sync458r	Berhasil	Berhasil
2	Pesan2.txt	sync458r	Berhasil	Berhasil
3	Pesan3.txt	sync458r	Berhasil	Berhasil

Table 4.11 Pengujian Fungsionalitas Steganografi LSB

Proses Steganografi				
No.	Nama Citra	Nama Berkas	Penyisipan	Penguraian
1	Peppers.bmp	Pesan1.txt	Berhasil	Berhasil
2		Pesan2.txt	Berhasil	Berhasil
3		Pesan 3.txt	Berhasil	Berhasil
4	Baboon.bmp	Pesan 1.txt	Berhasil	Berhasil
5		Pesan2.txt	Berhasil	Berhasil
6		Pesan3.txt	Berhasil	Berhasil
7	Lena.bmp	Pesan1.txt	Berhasil	Berhasil
8		Pesan2.txt	Berhasil	Berhasil
9		Pesan3.txt	Berhasil	Berhasil

4.4.2.2 Hasil Pengujian Kinerja Perangkat Lunak

Pengujian kinerja perangkat lunak terbagi atas pengujian kinerja kode Huffman, kinerja kriptografi DES, serta kinerja dari steganografi LSB. Pada pengujian kinerja kode Huffman, dilakukan perhitungan terhadap faktor dan rasio kompresi. Hasil dari proses kompresi yang disajikan dalam aplikasi ini adalah kode Huffmannya. Tabel 4.12 menunjukkan kinerja dari proses kode Huffman.

Table 4.12 Pengujian Kinerja Kode Huffman

Proses Kode Huffman					
No.	Nama Berkas	Faktor	Rasio %	Kompresi	Dekompresi
1	Pesan1.txt	1,83	54,71	0101101110000 0001001010111 0001011110110 0000000110100 0001001101110 1100010100010 1111000100000 0010110001110 0100001110010 0110011001100 1110100010000 10011 (...)	Kriptografi, secara umum adalah ilmu dan seni untuk menjaga kerahasiaan berita [bruce Schneier - Applied Cryptography]. Selain pengertian tersebut (...)
2	Pesan2.txt	1,61	61,98	1101111101110 0001000011110 0111000001111 1100001101010 0110011001010 1110110100110 0011100110100 1000001001001 1001010001100 1001111100010 010001 (...)	// put at first, the minimum level to display the line 1:To use one command, type it inside the message window where you usually type to chat. 20:@h (...)

3	Pesan3.txt	1,58	63,49	0101100101100 1011001011001 0110010110010 1100101100101 1001011001011 0010110010110 0101100101100 1011001011001 0110010110010 1100101100101 1001011001011 00101 (...)	========== ========== === ### OpenKore what-will-become-2.1 ========== ========== === Features: - New (...)

Tampak pada tabel 4.6, Pesan1.txt memilki rasio kompresi yang paling kecil. Pada kompresi Huffman, nilai dari rasio kompresi yang kecil diakibatkan karena banyaknya simbol yang sama dalam suatu berkas. Dengan semakin kecilnya rasio, maka semakin bagus kualitas kompresinya.

Pengujian kinerja terhadap kriptografi DES, pesan yang diuji menggunakan kunci 'sync458r'. Plainteks yang menjadi masukkannya adalah kode Huffman dari proses kompresi. Hasil dari proses kriptografi berupa cipherteks. Proses dari pengujian kriptografi DES ditunjukkan pada tabel 4.13.

Table 4.13 Pengujian Kinerja Kriptografi DES

Proses Kriptografi				
No.	Nama Berkas	Kunci	Enkripsi	Dekripsi
1	Pesan1.txt	sync458r	âÕ¥"šûŽ N|- Æ0¸4if›ˆ~¹I£%|- óGšÞu×xŠ98vv[Æ Ú÷R¥Œ·7¥_Ì®X6ï¸ .i¡xo.æ::ëEªÙgYê´y Q"'.fªŒŸšáìµc¿%þ 6±?6G»Ì;ÂÄ¨™rÝ Ô • luêp€ ,»AÃ•ZÙ ¥f{‚‰ÿ1. (...)	01011011100000001 00101011100010111 10110000000011010 00001001101110110 00101000101111000 10000000101100011 10010000111001001 10011001100111010 001000010011 (...)

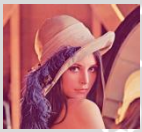
2	Pesan2.txt	sync458r	5“Å1Ã/|ƒè>HF)4T M'\QTæÈÍWÄeÙHj „õ\`cšÎL‡ 0ê¬ƒryA^{a}rµ□˜¸®a¡ ¢)éòÄáã¯o@5U=F ËTûà.½lü÷ XâþÐJ* õ6È·•OhÌbíñCEþR,, XÜv~aùÏß (...)	11011111011100001 00001111001110000 01111110000110101 00110011001010111 01101001100011100 11010010000010010 01100101000110010 01111100010010001 000100100011 (...)
3	Pesan3.txt	sync458r	¤?q • ß+óóù^?ŒÓ^{a}± QyO·¬_È□™ò@u¿ M ÿï¥5ù˜!«Ÿ´FN1æ= Q~¢(í ¬œÖr Ìx÷V_MQÎ$e“z¬È° @õþ`uì)Ë[hH"o_Î…Xü¿'æ Ìƒ¶bµdù!Ž%û (...)	01011001011001011 00101100101100101 10010110010110010 11001011001011001 01100101100101100 10110010110010110 01011001011001011 00101100101100101 100101100101 (...)

DES mengenkripsi plainteks setiap blok 64-bit, karena plainteksnya berupa kode Huffman maka hal ini tentu saja ada sedikit perubahan. Misalkan saja hasil dari kode Huffman panjangnya 30-bit, maka untuk masuk proses enkripsi DES harus ditambahai 34-bit lagi. Dalam aplikasi ini penambahan bit berupa bit ‘0’ sebanyak *n* sampai kelipatan 64-bit terdekat.

Proses dekripsi DES memerlukan kunci yang sama dengan proses enkripsinya. Jika kunci yang digunakan pada dekripsi tidak sama, maka hasil plainteksnya akan berantakan dan tidak bisa dibaca. Pada tabel 4.7, hasil dari proses dekripsi bukan berupa plainteks melainkan kode ‘1’ dan ‘0’. Sebenarnya kode ‘1’ dan ‘0’ ini merupakan kode Huffman sehingga untuk membaca plainteks yang utuh diperlukan proses lebih lanjut yakni dekompresi Huffman.

Pengujian kinerja berikutnya dilakukan pada steganografi LSB. Dalam pengujian ini melibatkan perhitungan MSE dan PSNR. Hal ini penting karena untuk menguji seberapa besar pengaruh derau yang disisipkan terhadap hasil citra stego yang dihasilkan. Pada tabel 4.14 ditunjukkan pengujian dari kinerja steganografi LSB.

Table 4.14 Pengujian Kinerja Steganografi LSB

Proses Steganografi					
No.	Nama Citra	Nama Berkas	Citra Stego	MSE	PSNR
1	 Peppers.bmp	Pesan1.txt	Peppers-stego1.bmp	0,0727	59,5135
2		Pesan2.txt	Peppers-stego2.bmp	0,1429	56,5798
3		Pesan3.txt	Peppers-stego3.bmp	0,4642	51,464
4	 Baboon.bmp	Pesan1.txt	Baboon-stego1.bmp	0,0181	65,5633
5		Pesan2.txt	Baboon-stego2.bmp	0,0356	62,6165
6		Pesan3.txt	Baboon-stego3.bmp	0,1166	57,4639
7	 Lena.bmp	Pesan1.txt	Lena-stego1.bmp	0,0046	71,5328
8		Pesan2.txt	Lena-stego2.bmp	0,0089	68,6129
9		Pesan3.txt	Lena-stego3.bmp	0,0290	63,5107

Dilihat dari tabel 4.14 nilai PSNR yang tinggi dicapai oleh lena.bmp dengan penyisipan Pesan1.txt. Hal ini disebabkan karena lena.bmp memiliki ukuran piksel yang relatif besar 512x512 px dan pesan yang disisipkannya pun terbilang sedikit sekitar 13184-bit. Dengan nilai PSNR yang tinggi tersebut, maka dapat dikatakan kualitas dari citra stego hampir menyerupai citra penampungnya.

Dari ketiga citra penampung tersebut memiliki batas kapasitas penyimpanan pesan tersendiri yakni :

Peppers.bmp = 128*128*3 = 49152 bit

Baboon.bmp = 256*256*3 = 196608 bit

Lena.bmp = 5125*5125*3 = 786432 bit

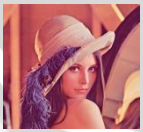
Oleh karena itu dilakukanlah pengujian pada berkas pesan setelah dikompres, apakah bisa disisipkan atau tidak. Pengujian batas maksimal pesan ditunjukkan pada tabel 4.15.

Table 4.15 Pengujian Batas Maksimal Pesan Yang Disisipkan

No.	Nama Berkas	Panjang Bit Sebelum Dikompres	Panjang Bit Setelah Dikompres
1	Pesan1.txt	13184	7213
2	Pesan2.txt	22656	14043
3	Pesan3.txt	71872	45631
4	Pesan4.txt	78720	49440
5	Pesan5.txt	125568	49418
6	Pesan6.txt	218752	139847

Seberapa cepat aplikasi ini melakukan proses penyisipan dan penguraian pun akan diuji. Pengujian ini bertujuan untuk mengetahui relatifitas waktu eksekusi terhadap pesan yang disisipkan serta citra penampungnya. Tabel 4.16 menunjukkan pengujian terhadap waktu eksekusi.

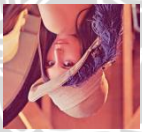
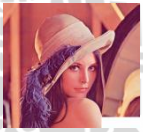
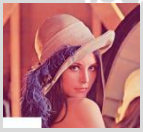
Table 4.16 Pengujian Kinerja Aplikasi Terhadap Waktu Eksekusi

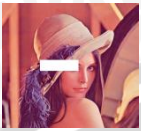
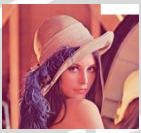
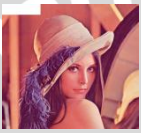
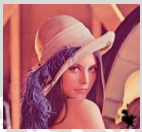
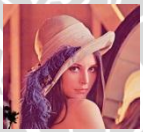
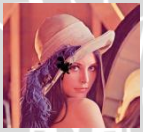
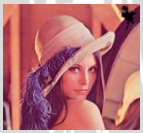
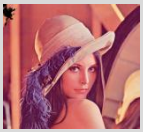
Proses Steganografi			
No.	Nama Citra	Nama Berkas	Waktu Eksekusi (detik)
1	 Peppers.bmp	Pesan1.txt	4
2		Pesan2.txt	8
3		Pesan3.txt	32
4	 Baboon.bmp	Pesan1.txt	3
5		Pesan2.txt	5
6		Pesan3.txt	33
7	 Lena.bmp	Pesan1.txt	4
8		Pesan2.txt	6
9		Pesan3.txt	32

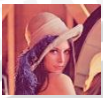
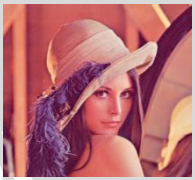
4.4.2.3 Hasil Pengujian Ketahanan Citra Stego

Pengujian terhadap ketahanan citra stego yang dihasilkan pada aplikasi ini perlu diuji untuk mengetahui seberapa bagus kualitasnya. Untuk proses pengujian ini, dilakukan manipulasi pada citra stego yakni dengan cara di rotasi, *cropping*, *noise*, dan *resize*. Sampel yang digunakan untuk pengujian citra stego diambil dari citra stego lena-stego.bmp yang merupakan hasil dari proses penyisipan lena.bmp dengan pesan1.txt. Pada tabel 4.17 ditunjukkan hasil dari pengujian ketahanan citra stego terhadap beberapa manipulasi citra.

Table 4.17 Pengujian Ketahanan Citra Stego

Manipulasi		Citra manipulasi	Hasil penguraian	Hasil
Rotation	90 cw		ab nltagma dfn-annamnutftdama miomneslkundedi ebtet/una (...)	Gagal
	180 cw		en rrlh gntmghe da unn nnatngenp,gl .(...)	Gagal
	270 cw		papnk sinnia eaa tnea ttdnbdu aidlreatuim gnpapeo (...)	Gagal
Croping 144 x 44 px	Bawah kanan		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil
	Bawah kiri		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil

	Tengah		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil
	Atas kanan		yang berhubungan denmgmatrmal arnta yotaentiarsst (...)	Gagal
	Atas kiri		mAuunssraht adaotaentiarsst adaotaentiarsst adaotaentiar (...)	Gagal
	Bawah kanan		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil
Noise	Bawah kiri		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil
	Tengah		Kriptografi, secara umum adalah ilmu dan seni untuk (...)	Berhasil
	Atas kanan		kerahasiaan data, keabsahal nseanrikas hpiurltnan ltnalcramanaun (...)	Gagal
	Atas kiri		,a mtpaannnf.nAtka tirgate m adalah ilmu dan seni (...)	Gagal

Resize	Pengecilan 256x256 px		nlenna ainirakln toll Hsl ueg.icnrieyei (...)	Gagal
	Perbesaran 768x768 px		jsprgateteuau eapnggun /u kg .dhm,puduagea k irgna (...)	Gagal

4.5 Pembahasan

Dari hasil uji coba yang dilakukan akan dianalisa lebih lanjut. Hal ini dilakukan untuk mengetahui apakah hasil yang sudah didapat sudah memenuhi tujuan dari penelitian ini. Selain hal tersebut, diambil kesimpulan dari hasil pengujian yang didapat.

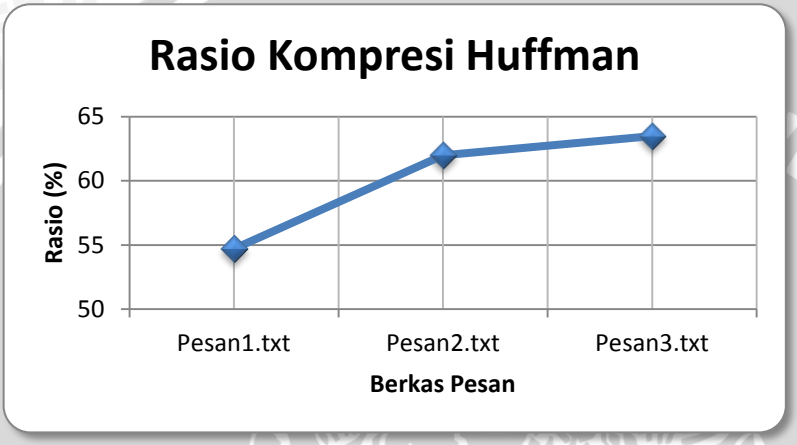
4.5.1 Analisa Hasil Uji Fungsionalitas Perangkat Lunak

Hasil yang diperoleh pada pengujian fungsionalitas dapat dikatakan sudah memenuhi kebutuhan dari perangkat lunak. Bisa dilihat dari tingkat keberhasilan perangkat lunak dalam melakukan proses penyisipan maupun proses pengurain. Dalam proses penyisipan sendiri, pesan mampu dikompres dengan baik. Hasil pengompresan ini diamankan dengan enkripsi dan disisipkan ke citra penampung sehingga menghasilkan citra stego yang kualitasnya sama dengan citra penampungnya. Untuk proses penguraiannya pun tidak mengalami kegagalan, sehingga pesan yang telah disisipkan dapat diungkap kembali menjadi teks yang utuh.

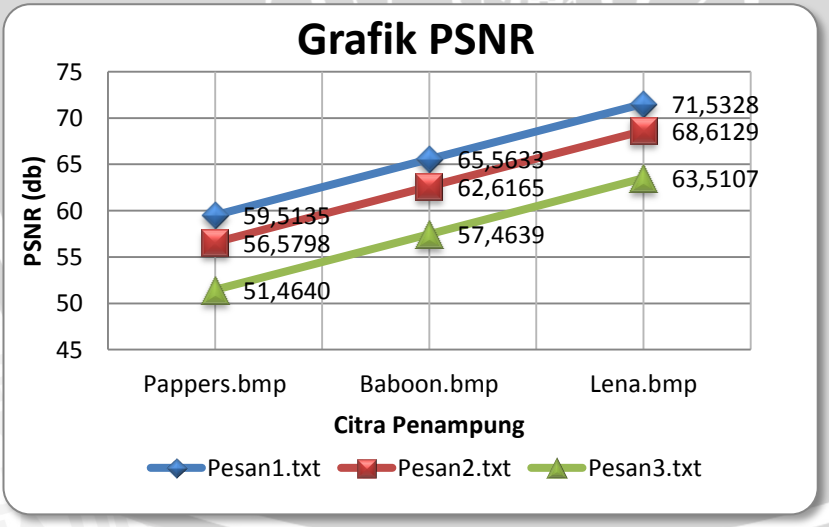
4.5.2 Analisa Hasil Uji Kinerja Perangkat Lunak

Berdasarkan hasil pengujian kinerja perangkat lunak pada proses kompresi dapat disimpulkan cukup bagus. Nilai rasio komrpresi pada ketiga berkas yang dikompres rata-rata sekitar 60,06%. Dengan rasio tersebut berarti pesan mampu dikompres 39,94%.. Berkas yang dikompresi pada kode Huffman dipengaruhi

oleh banyaknya jumlah simbol pada berkas tersebut. Semakin banyak jumlah simbol yang sama dalam suatu berkas, maka akan menghasilkan rasio kompresi yang kecil. Itulah sebabnya kode Huffman baik untuk mengkompresi sebuah teks. Pada gambar 4.4 ditunjukkan grafik dari rasio kompresi terhadap ketiga berkas uji.



Gambar 4.4 Grafik Rasio Kompresi Huffman

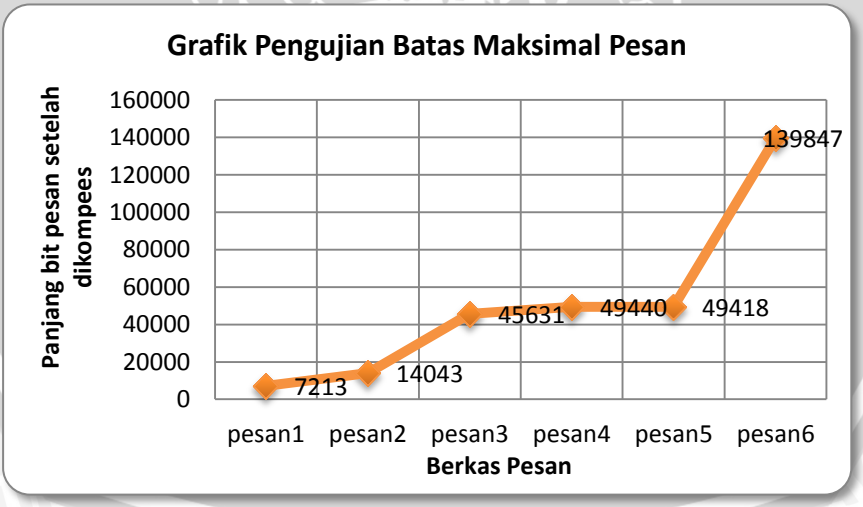


Gambar 4.5 Grafik PSNR

Grafik dari nilai PSNR ditunjukkan pada gambar 4.5. Berdasarkan grafik tersebut, dapat dikatakan bahwa kualitas citra stego yang dihasilkan dari perangkat lunak ini cukup bagus. Hal ini dapat dilihat dari nilai PSNR yang rata-rata di atas 50db. Rata-rata nilai PSNR yang diperoleh dari citra Pappers.bmp 65,54db, citra Baboon.bmp 62,60db, dan citra Lena.bmp 57,48db.

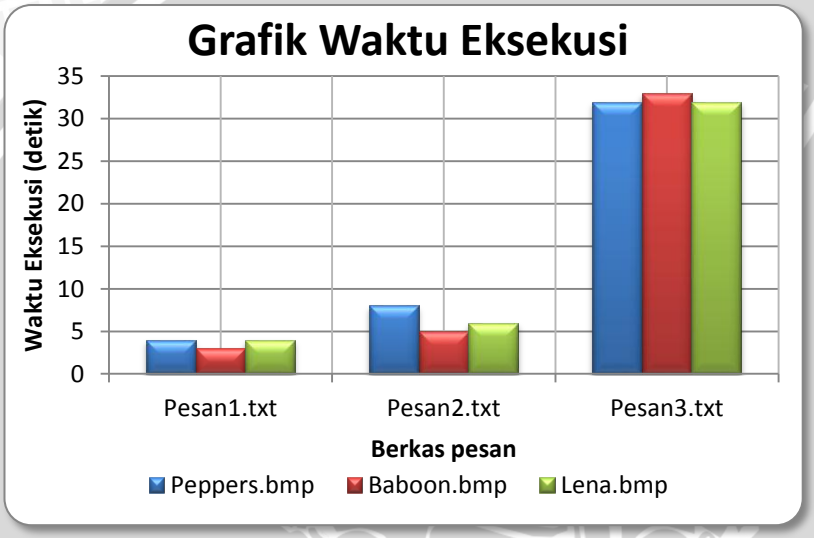
Hasil analisa dari grafik 4.5 menunjukan bahwa nilai PSNR berbading lurus dengan pesan yang disisipkan dan citra penampungnya. Jika pesan yang disipkan besar, maka untuk menghasilkan PSNR yang bagus yaitu dengan memperbesar ukuran dari citra penampungnya. Begitu juga sebaliknya jika pesan yang disisipkan besar sedangkan citra penampungnya kecil maka akan dihasilkan nilai PSNR yang buruk, sehingga citra stego yang dihasilkan mudah dikenali.

Dari hasil pengujian batas maksimal pesan yang ditunjukkan pada grafik 4.6, maka dapat disimpulkan bahwa citra Peppers.bmp hanya mampu manampung pesan1.txt, pesan2.txt, dan pesan3.txt saja dikarenakan batas maksimal dari citra Pepers.bmp hanya 49152 bit. Sedangkan untuk Baboon.bmp dan Lena.bmp, mampu menampug keenam berkas tersebut karena kapasitas tampungnya jauh lebih besar dibandingkan dengan hasil kompresi pesannya.



Gambar 4.6 Grafik Maksimal Pesan

Hal lain yang perlu diperhatikan untuk menganalisa perangkat lunak adalah waktu proses eksekusi. Gambar 4.7 menunjukkan waktu dari proses eksekusi penyisipan dan penguraian. Waktu eksekusi selalu berbanding lurus dengan jumlah pesan yang disisipkan. Semakin banyak pesan yang disisipkan, maka proses eksekusi akan semakin lama begitu juga sebaliknya.



Gambar 4.7 Grafik Waktu Eksekusi

4.5.3 Analisa Hasil Uji Ketahanan Citra Stego

Menurut hasil uji ketahanan citra stego yang telah dihasilkan pada tabel 4.17 dapat disimpulkan bahwa proses penguraian akan gagal jika citra stego dilakukan manipulasi rotasi serta *resize*. Perubahan letak bit-bit pada proses rotasi menyebabkan susunan dari bit pesan yang disisipkan berubah. Proses *resize* menyebabkan bit-bit pesan hilang. Dengan hilangnya susunan bit-bit pesan ini tentu saja menyebabkan pesan tidak dapat dibaca menjadi informasi yang utuh.

Pada pengujian *cropping* terdapat dua kemungkinan yang dihasilkan yakni pesan dapat dibaca menjadi informati yang utuh atau hanya sebagian saja. Pesan dapat dibaca secara utuh jika dilakukan *cropping* bawah kiri, bawah kanan, tengah. Namun akan

diperoleh pesan yang sebagian bila dilakukan *cropping* pada atas kanan atau atas kiri. Hal ini dikarenakan proses penyisipan steganografi LSB yang dilakukan secara sekuensial mulai indek 0,0 pada citra penampung maka akan sangat rentan jika di-*cropping* pada bagian atas. Semakin banyak pesan yang disipkan pada citra penampung dengan ukuran yang kecil akan menjadikan citra stego rentan terhadap serangan *cropping* disegala titik.

Penambahan *noise* sebenarnya hampir sama dengan *cropping*. Penambahan *noise* dibeberapa titik tidak menyebabkan pesan menjadi rusak kecuali penambahan noise dibagian atas. Bagian atas kiri, atas tengah, atau atas kanan sangat rentan sekali jika dilakukan penambahan noise. Penambahan *noise* pada suatu titik menyebabkan susunan RGB citra stego berubah. Pada waktu dekripsi maka akan dihasilkan kode Huffman yang tidak sesuai dengan proses enkripsinya sehingga menghasilkan pesan yang tidak beraturan dan tidak bisa dibaca.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitaian yang telah dilakukan dapat diambil kesimpulan sebagai berikut :

1. Teknik pengamanan data dengan mengkombinasikan kode Huffman, kriptografi DES dan steganografi LSB dapat diimplementasikan secara bersamaan. Pesan yang disisipkan sebelumnya dikompres dengan kode Huffman sehingga menghasilkan jumlah bit yang lebih sedikit. Kode Huffman yang telah terbentuk akan diamankan dengan enkripsi DES dan hasil cipherteksnya disisipkan dengan LSB.
2. Dari hasil pengujian didapatkan citra stego yang relatif sama dengan citra penampungnya. Hal ini bisa dilihat dari rata-rata nilai PSNR dan MSE yang diperoleh yaitu sekitar 61,87db dan 0,108. Nilai PSNR dan MSE-nya dipengaruhi oleh seberapa banyak jumlah pesan yang disisipkan dan luas dari citra penampungnya.
3. Dari keempat manipulasi citra yang dilakukan ada beberapa yang gagal dan berhasil. Pada proses rotasi akan membuat pesan yang ada pada citra stego menjadi rusak. Begitu juga dengan proses *resize* yang akan menghilangkan beberapa bit dari pesan. Untuk proses *cropping* dan *noise*, pesan dapat dibaca kembali menjadi informasi yang utuh dengan syarat tidak dilakukan *cropping* atau penambahan *noise* pada bagian atas atau bagian dimana terdapat bit-bit dari pesan tersebut berada.

5.2 Saran

Beberapa saran yang diperlukan untuk mengembangkan penelitian ini lebih lanjut sebagai berikut :

1. Jumlah pesan dan banyaknya simbol yang sama pada teks yang disisipkan sangat mempengaruhi nilai dari rasio kompresi dan PSNR. Dengan memilih metode kompresi yang cocok pada suatu jenis berkas akan mampu meningkatkan PSNR. Semakin kecil rasio kompresi, maka nilai PSNR akan tinggi.
2. Peningkatan keamanan pada kriptografinya bisa ditingkatkan dengan kriptografi asimetri agar tidak mudah dibongkar.

DAFTAR PUSTAKA

Burger, W., dan Burge, M. J. 2007. *Digital Image Processing, An Algorthmic Introduction Using Java*. Springer. Washington DC.

Cox, I., Miller, M., Bloom, J., Fridrich, J., dan Kalker, T. 2007. *Digital Watermarking and Steganography*. Morgan Kaufmann.

Dent, A. W., dan Mitchell, C. J. 2005. *Cryptography and Standards*. Artech House. Boston.

Elfirman, M. Z. 2010. *Steganografi Ciphertext Pada Citra Digital Menggunakan LSB*. Universitas Brawijaya. Malang.

Kusdinar, P., Wardoyo, I., dan Taufik, I. H. 2005. *Kompresi Teks dengan Menggunakan Algoritma Huffman*. Jurusan Teknik Elektro Institut Teknologi Bandung. Bandung.

Liddell, H. G., dan Scott, R. 1996. *Greek-English Lexicon : Ninth Edition with a Revised Supplement*. Oxford University Press. New York.

Mao, W. 2003. *Modern Cryptography: Theory and Practice*. Prentice Hall PTR.

Menezes, A., Oorschot, P. v., dan Vanstone, S. 1996. *Hand book of Applied Cryptography*. CRC Press.

Munir, R. 2004a. *Kuliah IF5054 Kriptografi : Data Encryption Standard (DES)*. Institut Teknologi Bandung. Bandung.

Munir, R. 2004b. *Kuliah IF5054 Kriptografi : Pengantar Kriptografi*. Institut Teknologi Bandung. Bandung.

Munir, R. 2004c. *Pengolahan Citra Digital dengan Pendekatan Algoritmik*. Informatika. Bandung.

Pu, I. M. 2006. *Fundamental Data Compression*. Butterworth-Heinemann. Great Britain.

Salomon, D., dan Motta, G. 2010. *Handbook of Data Compression*. Springers. Verlag London.

Sayood, K. 2006. *Introduction to Data Compression*. Morgan Kaufmann.

Sharma, R., Walia, E., dan Sharma, D. 2011. *Analysis of Non-Adaptive and Adaptive Edge Based LSB Steganography for Colored Images*. International Journal of Computing and Business Research.

Tilborg, H. C. A. v. 2005. *Encyclopedia of Cryptography and Security*. Springer. USA.